PRINCIPLES OF DESIGN IN THE OCTOPUS COMPUTER NETWORK*

John G. FLETCHER

University of California, Lawrence Livermore Laboratory,
Livermore, California 94550, U.S.A.

The concepts and methods are reviewed which have proven to be of the most value in designing and implementing the Octopus computer network, which is one of the largest concentrations of computing capability in the world. The discussion summarizes design principles relating to the scope of system software, privacy and security, processor organization, file structure, network connections, hardware selection, programming techniques, standards, and use of resources. Differences with what appear to be widespread belief and practice are cited, including such matters as the size of the programmer staff, the significance of the privacy issue, the importance of the choice of languages and language constructions, the primary causes of system failure, and efficiency.

1. INTRODUCTION

The Octopus computer network, designed and implemented at the Lawrence Livermore Laboratory of the University of California (LLL), constitutes one of the largest (if not the largest) concentrations of computing power and storage capacity in the world[1]. The network currently interconnects four C.D.C. 7600 computers; two C.D.C. STAR-100 computers are being added at the present time. On-line storage includes the 10^{12}-bit I.B.M. Photodigital store, while off-line storage is concentrated on over 30,000 magnetic tapes. Input-output capability includes two 18,000 line/minute printers, two I.I.I. FR-80 microfilm recorders, Evans and Sutherland LDS-1 displays, over 200 television monitors, and over 750 teletypewriters. The network has been in a continual state of change and growth for nearly a decade, new hardware being added as technology advances; as a consequence the system usually has included the fastest and largest capacity components currently available.

Octopus supports over 1500 interactive and batch users, and it is not uncommon to find 250 of them logged into the system at one time. The programs executed for these users range from highly interactive text editors and file query routines to huge numerical simulations which can execute for over ten hours on a C.D.C. 7600 and generate output filling 20 or more reels of magnetic tape. A typical daytime hour will see tens of thousands of messages pass through the network, including over 500 log in messages.

The preceding brief summary should make it clear that Octopus constitutes a very significant computer system implementation. It would seem therefore that any observations made or conclusions drawn about network and operating system design and implementation, which derive from experience with Octopus, should be of general interest. This is particularly so since those observations and conclusions are not always in agreement with widely accepted views and practice. The remainder of this presentation will summarize several principles based on Octopus experience. Since it is of course not true that all persons who work on Octopus share identical views, the statements made here under editorial we at best represent a consensus and of necessity reflect personal opinions of the present author.

*Work performed under the auspices of the Energy Research and Development Administration.

2. SYSTEM FUNCTIONS

Probably the most fundamental principle of all is that the system be kept as small as possible. System is here defined as the set of routines which have full access to the capabilities of the processor hardware, i.e., what is often called the kernel of the system. In the case of computers that execute user programs (which are called workers in Octopus terminology and include the large C.D.C. computers listed earlier), system routines execute in a privileged mode not available to users' programs. The system should perform those functions, and only those functions, which if a user were allowed to perform them would permit him to damage the system, to disturb other users, or to gain special privilege for himself. The system performs file accesses, controls the flow of messages through the network, allocates resources, carries out all I/O activity, interprets requests made by user programs, and handles all matters relating to accounting and security. The system does not include utility routines (compilers, interpreters, relocatable loaders, text editors, information retrievers, debuggers, etc.) nor of course does it include any application programs.

The importance of thus limiting the system is that it keeps the network design manageable. Because of the size of Octopus the system cannot be small, but if it included unnecessary functions it would become too large to keep well organized. Having a system with limited functions also allows the programming load to be distributed more widely. For example, the responsibility for programming a specialized compiler or other routine of interest only to a small group of users can reasonably be placed on that group and not on a central staff.

The effect of this principle is that the system programming staff at LLL has over the years averaged about 30 persons. This figure includes those who program the most widely used compilers (e.g., Fortran, Algol, Cobol) and other utility routines as well as those who program the system proper. Further, this staff is responsible not only for maintaining and modifying the system, it has designed the entire network from scratch. (Octopus does not employ manufacturers' or other commercial software, a primary reason being that that software neither adheres to the principles outlined here not provides for network activity, and is therefore inadequate for LLL needs[2]). It is so clear from the Octopus experience that one or two persons are

sufficient to program a minicomputer and that no more than half a dozen are needed for a worker computer operating system or a major compiler that we find it difficult to imagine how tens or hundreds of persons are kept busy on similar projects at other installations.

3. PRIVACY

It is really the issue of privacy that dictates which functions are assigned to the system. It is undesirable that the carelessness (or maliciousness) of one user interfere with the activity of another user or allow unauthorized access to that user's private data base. All system routines must be written with the intent of assuring privacy; we have found that this is not difficult to do. Except for the questions of programmer oversights and careless implementation, Octopus software appears to be entirely secure. A good analogy that has been suggested in regard to this is that it is not difficult to build a leak-proof container for liquid if one uses solid plates and not wire mesh in the design. The analogy suggests, moveover, that it may not be easy to fix the leaks in an improperly designed and already implemented system. The surprising thing is that so many commercially available systems are built of wire mesh.

Briefly, Octopus uses a combination (or password) to verify that a user logging in is in fact who he claims to be. Thereafter, the system relies on records associated with that user to determine which files and other resources may be made available to him.

4. SYSTEM ORGANIZATION

The network is fully interactive, the worker computers being time-shared and multiprogrammed. The vastly reduced turnaround time obtained by such operation very significantly improves efficiency from the users' viewpoint. Batch-like operation is included as a special case.

Processor hardware permitting memory partitioning and automatic program relocation is essential. Only with such hardware can the users be permitted to program freely in whatever language they choose and still execute their programs at full processor speed. Denying such capability would constitute an intolerable burden on the users. More advanced memory hardware (such as two-segment relocation or paging) which permits reentrant programs and other forms of memory sharing has been found to be a worthwhile improvement over simpler schemes, since more efficient utilization of memory is possible.

The central Octopus filing system, called Elephant, which uses the 10^{12}-bit Photostore[3], is accessed through a system of directories[4]. A directory is a special kind of file which associates pointers to other files with symbolic names for those files; the files so listed may be either simple (data or program) files or other directories. The entire directory structure is of the form of a general directed graph. A user can access only that portion of the structure accessible by a chain of pointers starting at a root directory associated with him. This scheme has proven a considerable success, particularly since it provides the opportunity for very general information sharing arrangements among users, as well as permitting each user to logically structure his data in a convenient way.

5. NETWORK ORGANIZATION

Octopus is currently organized as a superposition of partially independent subnetworks that use store and forward protocol. Each subnetwork is centered on a minicomputer, the concentrator of the subnetwork, which is connected to all the worker computers and to whatever other gear is appropriate to the function of that subnetwork. Ideally, each subnetwork has a single function such as central file storage, teletypewriter message handling, remote job entry, etc. This organization permits new facilities to be added to Octopus or existing facilities to be modified with only minimal disruption of activities unrelated to the change. A highly centralized network, such as the original Octopus design which placed all non-worker functions into a single concentrator or head of the network, suffers because continual growth requires continual hardware and software modification of the head. This causes the head to be frequently unreliable or inoperative, and when the head is not functioning properly the entire network is unusable. The present trend is toward even greater decentralization: Eventually the network will consist of a number of computers, each with its own functions such as the worker function, file storage, message switching, etc. A pair of computers will be directly connected wherever necessary as dictated by data traffic load.

Octopus is not designed around a particular manufacture of hardware and in fact includes computers and other hardware from many different manufacturers. Only with this freedom can the network be assured of including the most cost-effective equipment as the system grows in response to changing technology. It is necessary for LLL to employ an engineering staff which designs and implements the interfaces between the various pieces of hardware. This has had the benefit that there can be close interaction between the hardware designers and the system programmers who will use the hardware, resulting in designs which represent excellent compromises between hardware complexity and software overhead, a situation which unfortunately does not obtain for many commercially available devices.

It might be worth voicing a public complaint about how poorly many manufacturers understand the ways in which their own products are used and about the design shortcomings that thereby result. Many devices are such that it is difficult or impossible to insulate users and assure privacy; for example, multiterminal output display systems often have terminal selection instructions embedded in their display lists, thus requiring the system software to examine and censor all lists - a considerable overhead. Other devices lack obviously important status registers; for example, there is a paged processor that does not provide the offending virtual address at the time of a page fault, thus requiring interpretive reexecution of the trapped instruction (which, moreover, is not always possible). Many large storage devices have unacceptably small random access times, since they combine very slow mechanical fetching mechanisms with negligible facilities for overlapping operations[5].

A related problem is that of useful hardware that is not manufactured at all. For example, because of the large amounts of data involved, Octopus must use rotating storage for buffering information while it awaits its turn to move further through the network over a particular channel. Inefficiencies arise because of delays while read/write heads are moved back and forth between the area being written and the area being read. A disk that could be simultaneously read and written via two independently moveable heads would seem to constitute a reasonable and economic solution to the problem.

6. PROGRAMMING TECHNIQUES

Defensive programming seems to be the most important programming technique in Octopus. No computer in the network should rely on data received from

another computer to the extent that errors in that data could affect activities unrelated to the data or could cause the computer to malfunction. The same rule should apply to some extent between programming modules in a single computer, particularly when the modules are written by different persons. A computer being out of service should inhibit only those functions for which that computer is essential.

There are other programming concepts which seem to receive an inordinate amount of attention in the programming literature and should therefore be mentioned chiefly because they are only partially applied, or are not applied at all, in Octopus. Octopus system programs are written either in assembly language (more frequently without macros than with them) or in an LLL-designed, Fortran-derived language called LLLTRAN[6]. The system programmers tend to divide somewhat sharply into advocates of the two classes of languages, and a consensus does not exist. It seems fair to say, however, that there is no objective evidence that the programs written in the higher-level and lower-level languages differ significantly in logical quality, understandability, speed of programming, or adaptability to new situations; it is mainly a question of what is familiar to a given programmer.

It might seem reasonable to suppose that, if the entire system were written in a single language, there would be less difficulty in transferring a program from one programmer to another. This view of course is based on the dubious assumption that programmers can be skilled in only one language. Moreover, at LLL, there is relatively little transferring of programs because there is relatively system programmer turnover. This situation, which apparently does not obtain at most installations, is probably not unrelated to the LLL policy of offering considerable programmer responsibility and freedom (such as in the choosing of the language to be used). There is no two-level structure of system analysts and programmers. A single person, working closely with the small group involved in the same project, designs and implements a given program or routine. All that is required is adherence to a few simple standard Octopus conventions and protocols.

There is no widespread feeling among Octopus programmers that the "go to" construction is a source of difficulty or that every program must be block-structured. The problem of deadlocks (in which two interrelated activities await interminably for one another to proceed) has rarely manifested itself and, when it has, has been quickly discovered and resolved. Little, if any, interest has been aroused by the possibility of "proving" that the system is "correct." Such an effort would appear to be prohibitively time-consuming and would involve proofs whose own correctness would be suspect. The best guarantees of good software seem to be careful original design, conscientious observation during use, and prompt debugging. Experience has shown, moreover, that hardware failure is a greater problem than software failure. Both are subject to design flaws, but software once fixed remains fixed, while hardware continues to fatigue and wear out.

Since the Octopus programmers are present at the site where their system is used and can therefore observe the running system firsthand, little simulation is done. It is much easier and more certain to simply try out a new idea, preferably in a limited way (e.g., on one worker). It is important that a thorough effort be made to foresee and provide for all possible anomalies that may occur when a new program is executed but it has been found unprofitable to include in the initial design detailed automated responses to all unusual conditions, since it is very difficult to predict which combinations of unusual conditions will arise in practice. For example, if a processor proves so reliable that it seldom if ever experiences a hardware failure requiring a complete deadstart, it is not worth the effort required to make the deadstart soft (i.e., such that users experience minimal discomfort). On the other hand, if hardware failure proves to be a serious matter, it then becomes important to provide software defenses against its consequences. We have found that nearly all hardware failures result in highly erratic behavior that is readily apparent to users and operators. Quick observation of failures is enhanced by software-generated messages when anomalies are noted. We have not found, however, that it is worthwhile to design elaborate defenses against particular consequences of failure; we do not, for example, make all decisions relating to critical matters in two independent places.

Our own observation as to the area which must receive the most attention when designing a network is the matter of net effective data transfer rates. It appears that all components eventually become overloaded and must be augmented or replaced. The issue which should most dominate a designer's thinking is arranging efficient use of available capacity so as to prolong the life of the parts of the system.

7. STANDARDS

Octopus network standards have been selected by a learning process. The first time a type of facility is put into the network, protocols and formats are designed ad hoc by the engineers and programmers. Then, whenever it becomes clear that several facilities are being developed that have common characteristics, a standard is adopted to satisfy them all (as well as any foreseeable future developments), even if this means reworking some software already implemented. Any redundant effort implied by this approach is more than compensated for by the fact that design is not hampered nor degraded by adherence to standards adopted before the real nature of the problem had been observed. This principle, as well as several others noted previously, may be summarized by stating that Octopus uses an experimental approach. We do not adopt nor even define general principles before we have actually had experience with the situations to which they apply. This approach is of course made easier because of reliance on locally designed software and, to some extent, hardware, which are readily modifiable.

There are Octopus standards for interface protocols, message headings, file formats, command languages, and many other matters. One standard may be of particular interest: All network messages consisting of characters are in ASCII embedded in 8-bit bytes, even though no large computer prior to the STAR-100 is oriented toward 8-bit units. This is an example of setting a standard based on an estimate as to what most computers will be like in years to come. In many Octopus computers, the ASCII is repacked (with suitable escape conventions) into 7- or 6-bit fields, whatever is most convenient to the word-size and instruction set. (The ASCII is never converted to EBCDIC or other code, which would seem to be a complete waste of effort.)

8. RESOURCES

Probably the most important observation about Octopus is that it works; it improves the speed, efficiency, and quality of computation at LLL and thereby of most activity at LLL. This does not mean that all users of the system are fully satisfied, the reason being that, for Octopus as for all systems, resources are limited. The large hardware inventory summarized at the beginning of this article does not imply that Octopus users have access to practically unlimited computing power and storage capacity. The hard fact is that even - no,

especially - at large installations it is economically unsound to supply computing capacity much in excess of actual need. Those who suggest that modern computers have such a surfeit of resources that computational efficiency can be indiscriminately sacrificed to achieve allegedly higher purposes are not fully in touch with all practical realities.

REFERENCES

[1] J. G. Fletcher, The Octopus computer network, Datamation, vol. 19, no. 4, April 1973, 58-63.

[2] J. E. Requa, In-house vs. vendor-supplied software: a case study at Lawrence Livermore Laboratory, Atomic Energy Systems, Operations and Programming Association Meeting, October 1972.

[3] J. D. Kuehler and H. R. Kerby, A photodigital mass storage system, Proc. AFIPS FJCC, vol. 29, November 1966, 733-742.

[4] R. C. Daley and P. G. Neumann, A general purpose filing system for secondary storage, Proc. AFIPS FJCC, vol. 27, November 1965, 213-229.

[5] G. B. Houston, Trillion Bit Memories, Datamation, vol. 19, no. 10, October 1973, 52-58.

[6] S. F. Mendicino, R. A. Hughes, J. T. Martin, F. H. Mc Mahon, J. E. Ranelletti, and R. G. Zwakenberg, The LRLtran Compiler, Comm. A.C.M., vol. 11, no. 11, November 1968, 747-755.