OPERATING SYSTEM DESIGN WITH COMPUTER NETWORK COMMUNICATION PROTOCOLS

Wilfried G. PROBST
Dept. of Computer Science,
Concordia University

Gregor V. BOCHMANN
Dép. d'informatique et de recherche opérationnelle,
Université de Montréal,

Montréal, Québec, Canada.

In view of the size and complexity of modern operating systems, this paper proposes their subdivision into a set of smaller functional modules and the implementation of a number of their functions on separate hardware processors. The information transfer requirements resulting from physically separated system components are examined and the adaptation of a standard end-to-end protocol is suggested for an efficient solution to the interprocess communication problems.

1. INTRODUCTION

The last 25 years have witnessed rapid progress in electronics and computer hardware, with drastic improvements in the areas of computational speed, information storage volume and input-output (I/O) capabilities. Efficient utilization of these hardware resources would not have been possible, however, without a corresponding development of software facilities, designed to allow shared user access to a computer complex and optimized throughput of their jobs.

Section 2 of this paper briefly covers the evolution of these system facilities and describes the basic structure and components most commonly encountered in modern operating systems (OS). It shows not only their continuous expansion in capacity and flexibility, but also their growth in complexity and size which has led to increasingly large requirements of processor time and memory space. While this overhead may still be within acceptable limits on large scale computers, it becomes really serious on small mini-computers which are enjoying a widespread popularity nowadays due to their low cost/performance ratios.

Section 3 analyses various possible hardware and software solutions to this problem. It then proposes an approach which consists of dividing the normally main memory resident part along functional lines into a set of modules implementing a certain number of them on separate micro-processors and thus obtaining an OS where the basic software is distributed over distinct hardware units. In particular a major portion of all I/O processing routines are removed from the central processor (CP) and the resulting system becomes not only smaller and less complex, but it is also capable of an even higher degree of concurrency in most computer operations than a normal central memory resident one. The various advantages as well as the additional requirements of such an approach are discussed. While it will decrease the overhead in the central unit and improve the flexibility of information flow through the system, it also requires proper links between physically separated system components.

Section 4 deals with this aspect and indicates how general techniques developed for computer networks can be used for an efficient solution of the communication problems within a distributed OS. More specifically it is shown how protocols designed for virtual terminal connection and dialogue are quite suitable for adaptation to the requirements of properly controlled information flow within the system.

Finally, in order to illustrate all these concepts, it is shown how a relatively powerful OS could be implemented on a low-cost mini-computer system, using a number of micro-processors to handle most tasks associated with I/O operations, including file access, data transfer and peripheral device control. Possible connection of this system to a public data network is also discussed.

2. OPERATING SYSTEMS BACKGROUND

The basic purpose of any OS is to allow a group of people to share the use of a complex computer installation in an efficient manner, in order to maximize the throughput of their jobs[1]. Considering the capabilities as well as the cost of modern hardware, we are faced with the non-trivial task of operating the computer in such a way as to ensure optimal use of all its components and thus hopefully achieving the above-mentioned goal.

2.1 Historical Evolution

The limited capabilities and low speed of the first hardware generation did not permit much sharing of resources in the early days of computing. The entire computer installation was usually allocated to one user at a time, most operating procedures were done manually and OS software was practically non-existant.

The introduction of a number of improvements and new developments since the late 1950's increased computing power and speed by several orders of magnitude, but hardware complexity also reached a point where the skills of an average user were no longer sufficient to handle its operation efficiently. To mention only a few examples, computers were equipped with increasingly numerous and complex peripherals and I/O operations were performed by means of autonomous direct memory access channels (DMAC), capable of operating in parallel with the CP. Sophisticated interrupt systems enabled automatic detection of special conditions as well as synchronization between concurrent processes and, together with appropriate memory protection mechanisms, provided the necessary infrastructure for elaborate multiprogramming and time-sharing of hardware resources.

Standard software facilities were therefore introduced to provide the basic services required for the complicated task of operating the different units of a modern computer. For example, input/output control systems provided the necessary I/O functions and supervisors (monitors) handled interrupt processing, resource allocation and job scheduling. This software

constitutes the basis of any modern OS and the user communicates with it by means of an appropriate command language.

2.2 Basic Structure and Main Components:

In order to perform the various tasks outlined in the previous paragraphs, the classical single-processor OS has become a very large software system with a complex structure, incorporating many different components. Those performing supervisory functions which are called upon most frequently and whose prompt execution is essential (e.g. interrupt processing), form a nucleus that must reside in central memory (CM) at all times. Other less critical ones are normally placed on secondary storage to reduce CM overhead and are loaded only when required, overlaying routines which are no longer needed at that moment. The trade-off of this scheme, however, is an increase of CP time overhead, caused by those swapping operations.

The main parts of a typical OS can be briefly defined as follows[2], where we have emphasized the specific functions of those components which are more closely related to the following sections of this paper.

A. Executive Control Functions

This part is responsible for maintaining real-time execution control of the system environment, in particular:

(i) Job scheduling, resource allocation and event monitoring.

(ii) I/O control, including I/O scheduling, data transfer and device manipulation.

(iii) System communication, including operator console support and I/O queue maintenance (spooling).

(iv) Hardware error detection and recovery, program error control (overflows, etc.).

(v) Support for timing and debugging services.

(vi) Accounting procedures.

B. System Management Functions

This part contains the non real-time components of the OS, supporting both system and application programs by providing services such as OS management (e.g. system generation), program maintenance (incl. libraries and catalogues), compiler interfaces and support utilities.

C. Data Manipulation Functions

This third part of the OS allows the user to access and process data in general. To mention just a few examples, we usually find:

(i) File management facilities, including directories and user access control.

(ii) I/O support facilities for different data access modes (e.g. sequential) and record blocking.

(iii) File display and copy facilities.

(iv) Peripheral device support, including format conversion and data editing.

While there are several smaller yet successful special-purpose systems (e.g. dedicated to interactive timesharing), the approach most often taken by major computer manufacturers has been to build a single, large general-purpose system which offers a wide variety of services. Typical examples, such as OS/360 (IBM), have indeed become very large, specially if compilers and utilities are also included. The overhead created by these contemporary OS threatens to defeat the very purpose for which they were originally developed, namely a more efficient use of the hardware.

Firstly, it is not uncommon to see resident OS routines occupy more than 25% of the main memory, thus decreasing the amount available to users; on small systems this percentage can be even larger. Secondly, because of the numerous supervisory functions they have to perform, a substantial portion of total execution time is spent by those routines doing administrative work, while user tasks wait for the CP to become available. Finally, no matter how carefully programmed, they will inevitably be error-prone in view of their size alone.

A number of changes and innovations in hardware as well as OS structure and design have therefore been suggested, with the objective of improving overall performance by solving some of the problems outlined above. On the software side we have most notably the development of synchronization primitives at low levels (e.g. semaphores) as well as higher ones (e.g. mailboxes and monitors[1]). A number of programming techniques and tools are also being introduced, for example modular design, structured programming and even special languages suited for structured OS design (e.g., Concurrent PASCAL[3]).

In the next section we will briefly mention some other proposals involving different hardware utilization as well. We will then present a solution in which modular design techniques are carried over to the hardware implementation by assigning individual processors to different modules. This approach, particularly attractive in the case of smaller low-cost configurations, should prove to be generally useful in a wide range of applications such as systems with remote job entry or computers and terminals connected through public networks.

3. DISTRIBUTED OPERATING SYSTEMS

In order to support general-purpose systems, large and expensive computer mainframes were practically mandatory until recently. The increasing use of medium and large-scale integration techniques in electronic circuits has led not only to rapidly decreasing hardware costs, but also to the development of relatively inexpensive yet powerful minicomputers (minis for short) in the late 1960's. However, due to their limited memory size and usually less elaborate I/O and interrupt facilities, they often operate under control of a greatly simplified OS capable of just a few specific tasks (e.g. data acquisition, simple time-sharing or sequential batch only). Their growing popularity has therefore stimulated work in computer architecture as well as in OS design, with the objective of implementing more powerful and flexible software systems on mini-computer based hardware configurations.

3.1 Multiprocessing principles

One obvious solution to overcome the restrictions imposed by limited hardware capabilities consists of connecting several minis together and forming

a multi-processor complex. Current efforts in this area seem to be following two main directions, namely:

(i) Integration of several processors into one processing system, equivalent in computing power to much larger conventional machines; the PRIME project[4] and the C.mmp (HYDRA) system[5] are typical examples of this approach.

(ii) Connection of independant and not necessarily identical processing systems to share hardware and software resources; the DCS system[6], the MININET prototype[7] and the KOCOS complex[8] illustrate this direction.

A brief analysis of any OS, revealing the set of basic functions outlined in the previous section, would obviously suggest its decomposition into a group of modules, each with a specific purpose. Furthermore, it can also be seen that many of the tasks performed by these modules are quite independent of each other and could be performed concurrently. On a single processor true parallelism is of course impossible; instead the processor switches from one task to the other by means of intricate interrupt procedures, thus reducing delays to acceptable levels.

This modular OS design concept suggests therefore yet another way to increase the power of any system: reduce the CP overhead by freeing it of a number of tasks which do not require its considerable computing power and have them processed by separate hardware units. The CDC-6000 series[9] represents an early example of this design philosophy; certain functions (mostly I/O) are implemented on so-called "peripheral processors" each of which has a separate memory, enabling them to execute programs independently of the CP and each other. A common CM is used for communication and information exchange between all processors.

Recent hardware and software developments have enabled us to consider a more flexible and generalized extension of modular OS design into hardware architecture. In view of the rapid development of truly low-cost micro-processors (micros for short) since 1971, the practical implementation of a larger number of functional units on dedicated, autonomous processors is becoming economically feasible nowadays; the computational power required can be provided by one or more minis.

In such a distributed OS only those modules whose function and operational environment require the use of a CP, would remain in the central unit. Device controlling and I/O handling routines are the most obvious candidates for redistribution. There is indeed no valid reason requiring the CP to keep track of a variety of device-dependant details (e.g. specific channel commands and status bits) each time a program is simply requesting the transfer of a block of data between its buffer and a designated peripheral unit. Other routines which could quite logically be implemented outside the central unit, include major portions of the file management system (disc and tape controllers) and some parts of the job control functions, e.g. spooling and file transfers between devices without need for CP intervention.

Fig. 1 shows a hardware configuration, consisting of a TI-980B mini-computer system equipped with the peripherals usually needed for a meaningful system, on which experimental design work of distributed software is being carried out at present. The OS proposed for this configuration will be essentially file-oriented, similar in concept to those used in the above-mentioned CDC-6000 series (cf. SCOPE or

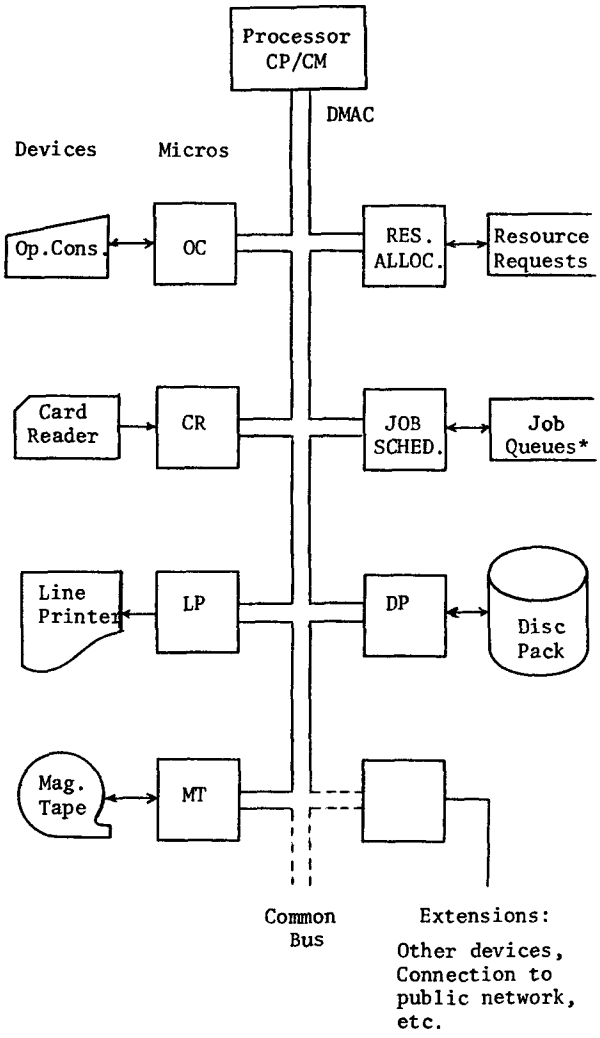
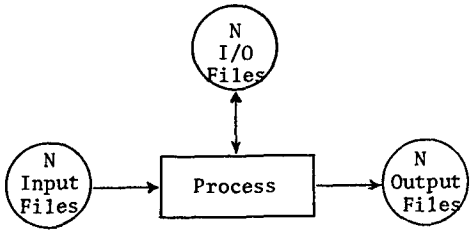


Fig. 1 * On DP or own floppy disc.

NOS[10]). As illustrated in Fig. 2, these files will be used in normal operation as input and/or output by several entities such as user tasks, utility routines, etc., called processes for short. The individual components of this system and the assignment of various functions to a number of micros are described in greater detail in the next section.



N = 0, 1, 2 ...

Fig. 2

3.2 Micro-Processor Functions

Each micro will handle the physical I/O operations for a particular device or type of device, taking care of the tasks normally performed by the device service routines in the OS supervisor. It will send a block of information from an internal buffer to a common bus (or to a network) in case of a system input and receive a block from the bus in case of output. Depending on the type of device it controls, the micro may also perform a number of addditional functions such as:

(i) Cardreader: Control card interpretation (to determine destination of input files); character code conversion.

(ii) Lineprinter: Code conversion; vertical and horizontal format control.

(iii) Job scheduler: Job input and output queue maintenance (batch jobs); control card interpretation (to identify user tasks); task scheduling and initialization.

(iv) Resource allocator: Process request queue maintenance; resource allocation; prevention of deadlocks.

(v) Magnetic tape: Label processing; blocking/deblocking of data records; tape positioning (e.g. rewind); handling of multiple files per reel.

(vi) Disc: On-line file management (incl. file directories); access control; file positioning (e.g. seek); blocking/deblocking; disc space allocation.

A typical device-micro implementation of these functions contains the basic components shown in Fig. 3 and requires each of them to be provided with a corresponding amount of private memory. The procedural and constant parts should be in read-only memory (ROM) for added protection.

As indicated above (see also Fig. 1), all micros are able to communicate directly with each other. This raises a number of problems the solution of which necessarily influenced the design of this system; we had to consider in particular the following:

- Proper routing of information over the common bus; this difficulty is overcome by means of proper bus design and the adoption of appropriate message formats and transmission protocols.
- Proper assignment of files to individual processes; the resource allocation micro controls the use of all peripheral system components in order to avoid conflicts and deadlock problems.

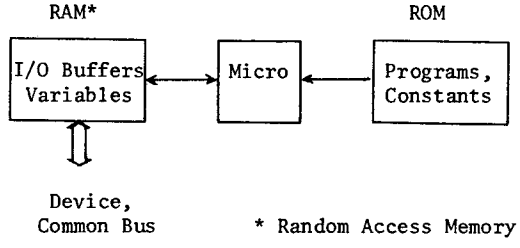


Fig. 3

3.3 Benefits and Drawbacks

Let us now examine some of the potential advantages offered by a system designed and implemented along these lines.

A. Reduced CP overhead

The part of the OS which still resides in the central unit requires a smaller amount of CM space and less frequent use of the CP, since it has fewer functions to perform than in a conventional system. This is particularly important in the case of a minicomputer configuration.

B. Flexible Information Flow

The flow of information through the system becomes far more flexible and efficient. In contrast to a traditional system where all I/O transfers involve the CM, it is now feasible to exchange data directly between any two peripheral units, e.g. card-to-disc or tape-to-printer transfers. A control card specifying source and destination files enables the OS to set up a direct transfer between the two corresponding micros. A batch job can be identified by a special JOB card recognized by the Cardreader which then sends it directly to the Scheduler and its job input queue. This design property therefore permits the implementation of a truly concurrent spooling system, eliminating unnecessary buffers and requiring only one passage through CM instead of the usual three for each job. Fig. 4 shows the normal path of a batch job through such a system and indicates several other possible data paths.

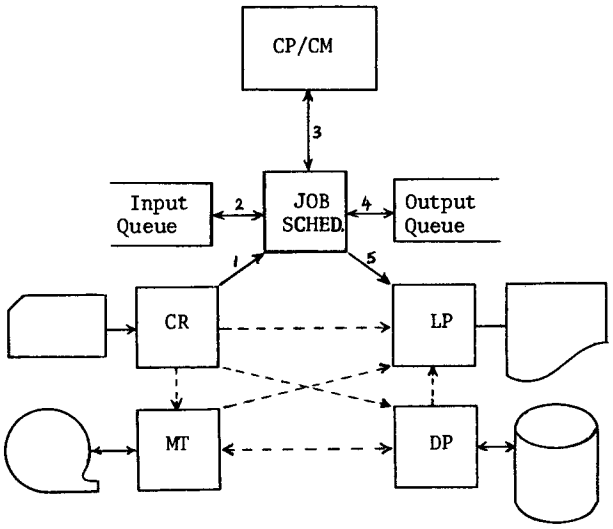


Fig. 4

C. Increased Protection

One of the threats affecting standard OS reliability is the possibility of accidental erasure of system components, tables or variables due to programming or design errors. Physical separation of modules and their implementation on different hardware processors will provide better protection against such flows and the use of ROM within the micros provides even further safeguards, as stated before. As far as privacy is concerned, additional security may be provided for

both transfer and storage of information, by including enciphering and deciphering mechanisms in the micros and allowing the user to specify his own secret encryption key[11]; the Cardreader might for example scramble all characters according to a key provided by the owner of the information, before it is sent further. Data thus stored could then only be retrieved and correctly interpreted if the key is also supplied.

D. Manufacturer-independent Device Selection

This system also makes it easier to connect peripheral units from different manufacturers to a given computer. Once a micro has been programmed to handle the particular device, the actual interfacing problems are greatly reduced as long as information transfers between system components are governed by a standard communication protocol. There would be no more need for introducing new device service routines into the CM resident portion of the OS and interfacing them with existing ones. Obviously, whenever a device is replaced by a different one, the micro in question must be reprogrammed.

Finally, let us now consider the two main disadvantages of such a distributed operating system.

E. Additional Hardware Requirements

When compared to a standard system, the principal hardware addition consists of the various micros and their memories. But with steadily decreasing hardware costs the above-mentioned benefits should be worth their price, which is even partially offset by a reduction in the amount of CM needed.

F. Communication Problems

As stated before, proper communication between distributed system components raises a number of problems not encountered in standard OS, where most interprocess exchanges are simply achieved either by means of shared variables in the CM or by passing address parameters (pointers) and where a subroutine call can be performed by a single CP instruction. In the next section we will look at these difficulties in greater detail and propose a solution inspired by similar problems occurring in computer network communication.

4. COMMUNICATION BETWEEN SYSTEM COMPONENTS

In the previous section we have discussed a few characteristics of a distributed OS; we shall now present the basic principles involved in its operation. Let us begin by looking at the file concept in greater detail.

There are two basic types of files, each with its own internal subdivisions, as follows:

(a) Text files, containing character information such as source programs and input data. The information in a text file is logically subdivided into:

- Pages, defined as a (variable sized) sequence of lines,
- Lines, defined as a variable sequence of characters.

A user job file, for example, could have the following traditional structure, where control cards are identified by a special character (e.g. !) .

```
! JOB   user and job information        (line 1)
! COPY  file-1   To   file-2             (line 2)
! FORTRAN                                (etc.)
      - - - - - - - - -
       Source program
      - - - - - - - - -
! EXECUTE
! EOF  (end of file)
```

(b) Binary files, containing bit strings or binary words in a machine-dependent format; they are normally produced by certain processes such as compilers and assemblers, i.e., object programs, or they may be created from the output of a user program. They are logically subdivided into binary records of variable length.

Both types of files may be accessed in basically two different ways, defined as follows:

(i) Sequential access. When the file is first built, its subdivisions (pages or records) are placed one after the other and may later be accessed only in the same sequence in which they were initially created, i.e., in order to get to record n of a binary file one must have accessed records 1,2,...,n-1 previously.

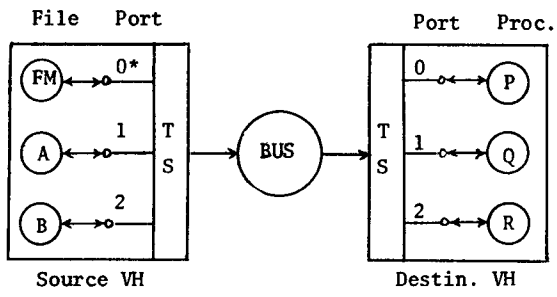
(ii) Direct access. Each subdivision is directly addressable and may be accessed independently from the others. This can be achieved, for example, by giving each record a number (relative to the first one) or a unique key which can be mapped into its address by using some kind of index table.

There may be some device dependant restrictions imposed on files, however; for example, files for printed output should not be binary (although octal or hexadecimal dumps might be of interest to some) and direct access files can only be implemented on certain types of storage devices.

The design of our distributed system allows some processes to share a single processor (multiprogramming), while others are implemented on separate fully dedicated ones. The most important problem to be solved, therefore, is to ensure correct and efficient communication between OS components, i.e. file access and transfer between micros in our case. In the next sections we will show how computer network communication protocols can advantageously be used for this purpose.

4.1 End-to-end Communication

Standard end-to-end protocols, implemented on top of the packet switching service in computer networks, have been proposed in order to provide users with a generalized interprocess communication facility[12,13,14]. They are based on the notion of collections of communicating processes, each such collection sharing a set of common resources and thus constituting a so-called "Virtual Host" (VH) which appears as a single entity to the packet switching network. Each of those processes is associated with a unique "port" number for communication purposes and all the ports in a VH are grouped together forming a "Transport Station" (TS) which provides port-to-port communication through the network. The introduction of the port concept as a network name space thus reduces the host-to-host protocol to a multiplexing of port-to-port, i.e. end-to-end protocols.



* Well-known port (File management)

Fig. 5

An analysis of the communications in the distributed OS described in the previous section reveals many similarities with computer network information flow. We may indeed consider the resources associated with each of the processors (CP and micros) to constitute a VH, and the common bus connecting them to form a simple network on which packets of information are sent from source hosts to destination hosts. Similarly, since each of those processors may contain one or more processes accessing different files, each corresponding VH will therefore require a TS, containing one port per active process or open file, in order to multiplex their access to the bus. This way, as an example, the reading of a file A by a process P can be accurately equated to an end-to-end transfer between two ports, as indicated in Fig. 5.

The packet switching facility carries information from a source TS to a destination TS, both of which are identified within the packet header by their addresses. Furthermore, since a TS is seen as a collection of ports from the communications side, specific port numbers will also have to be carried; together with certain other pieces of information (e.g. function codes) they would form the transport header which precedes the actual packet text[14]. Port numbers are locally associated with corresponding process or file names and while certain of these associations may be static (for some "well-known" system ports), most will have to be mapped dynamically by means of appropriate tables containing the names of currently active processes and open files. Although such mapping operations in the TS may seem unnecessarily complicated, they reduce substantially the amount of information required in the transport header (e.g. file user and owner names, etc.) and therefore simplify information transfer; the assignment of the port numbers during the establishment of an end-to-end communication link will be discussed in the next section (cf. SWITCH funtion).

The end-to-end protocol provides for transfer of letters between ports within the context of their association. Their size must be such that any physical record in the system can be placed in a letter, to avoid fragmentation of data seen by the process as belonging logically together. If the letter exceeds the packet size, it will be divided into fragments by the sending TS and reassembled upon arrival by the destination TS, to be delivered as a whole the way it was sent. The protocol ensures proper reassembly by means of letter reference and fragment numbers and should also include error and flow control at the letter level for a reliable operation of the OS. Very short letters, or telegrams, may be used in special cases to signal some unusual event or an interrupt (e.g. status checks, stop sending, etc.). While network protocols were designed for an environment where time delays and loss rates are much larger than in our bus-coupled multiprocessor environment, we feel that the overhead they represent may be reduced to an acceptable level (by some minor simplifications, if necessary) and should add substantially to the overall reliability of our OS. Further implementation details may be found in the afore-mentioned references[13,14].

4.2 File Access and Data Transfer

Connecting a user process to a file provides access to the information that the latter contains. Depending on its internal structure and the device on which it resides, this connection may however take different forms. We will therefore introduce the concept of Virtual File Access Protocol to define a standard file access method within the OS. This protocol should make all files look alike to the accessing processes, no matter where they reside. In most computer networks, whether existing or proposed, the basic transport services are enhanced by introducing additional facilities into the host-to-host protocol, such as those provided by the TELNET subsystem in the ARPA network[12]. This concept is also illutrated by the "Virtual Terminal Protocol"[15,16] and the "Bulk Transfer Function"[17], proposed to make a variety of terminals and file structures from different manufacturers look logically identical in the way they interact with the network. In view of the environmental similarities, these ideas can easily be adapted to our distributed OS. Just as the above-mentioned terminals or files require some form of intelligence to take care of their local handling, each individual peripheral unit in the distributed OS is handled by a micro-processor, programmed to transfer information according to the common file access protocol which has to provide the following functions:

A. Connection Control, containing a set of initialization directives for establishing connections between process and file entities, and for allowing such a connection to be changed; this liaison is established by exchanging identification, file characteristics and process requirement messages[16] and corresponds basically to an OPEN function.

In case of single file devices, e.g. card reader and line printer, there will only be one port in the VH and process-file linking is straight-forward. Multi-port hosts, such as disc and tape controllers handling several files concurrently, require a more elaborate connection procedure for efficient and flexible port assignment. The process P establishes initial contact with a well-known system port within the VH, which acts as a common entry point to the file management system, and sends his requirements, e.g. name of file and owner, access mode (read/write), etc. The VH uses a file catalogue to find it, verifies the legitimacy of the request (file protection) and then assigns a new port number to the file (see Fig. 5) by performing a SWITCH function such as the one described in[15]. This new number is returned to the calling process together with certain additional file characteristics, for example type (e.g. binary) and record size. All subsequent file-process communication will then go through this new port, until a CLOSE operation terminates the liaison and frees the port.

B. Dialogue Control, including the READ and WRITE functions. In case of sequential access, addressing of file components is relatively simple and consists of updating line and page pointers (text files) or record pointers (binary files) after each transfer. Useful positioning functions to be added for this case are BACKSPACE (decrease pointers by 1) and REWIND (reset pointers).

For direct access the addressing must be done by page or record designators which are mapped in the VH to the requested file component by means of an appropriate index containing the physical address of each directly accessible subdivision of that file. This function is performed by a SEEK operation prior to the actual READ or WRITE.

The actual data transfer occurs after the initial READ or WRITE request has been acknowledged by the micro to whom it was addressed; the "writer" will thereafter send one or more letters containing the requested data to the "reader", terminating with a special message indicating the end of transmission. One or more records and even whole files may be sent this way following a single READ/WRITE request.

D. Other Functions needed in the protocol are:

(i) The CREATE operation to establish a new file, requiring the host to allocate storage space and enter new names and addresses into the file directory together with other information (e.g. access restrictions) needed for proper file protection.

(ii) The DELETE operation to purge an existing file, erasing all references to it in the directory and freeing the storage space it occupied.

(iii) The CHANGE operation to modify a number of file parameters such as its name, protection mode, password, etc.

A number of implementations aspects, such as message formats, function codes and address size, are discussed at greater length in the aforementioned Virtual Terminal Protocols[15,16]. In view of their open-ended design, these proposals could be adopted for our purpose with only minor additions. Additional file ("bulk") transfer functions are presented in[17] and while we had to omit a number of operational details for the sake of brevity, the infrastructure presented in this paper should be quite adequate for a relatively powerful and flexible OS.

5. CONCLUSION

Since a conventional general-purpose OS is generally too large for implementation on a mini-computer system, we divide it into functional modules and implement a certain number of them on several independent micro-processors in order to reduce the overhead in the central unit. These micros are in particular responsible for controlling all peripheral device I/O operations and for transferring information between them and main memory. Communications are achieved by means of packet switching, and a higher level end-to-end protocol, providing for a number of file access functions, is also introduced.

This approach makes it feasible to develop a hardware system, based on low-cost components connected by a common bus, into a relatively powerful and flexible computing tool by means of an OS design emphasizing distribution of functions and concurrency of operations. The device controlling micros are powerful enough to emulate communication protocols similar to those found in computer networks. A properly designed bus structure, capable of message routing and multiplexing host-to-host communication over the bus (or time-sharing its use according to some priority scheme), should provide sufficient hardware capabilities for implementing proper packet switching transmission facilities.

The approach is also applicable to larger installations (to reduce the overhead in the central unit), systems with remote job entry facilities or computers connected to a private or public data network. In this case the lower level packet switching facilities would of course be provided by the network and the end-to-end protocol would be built on top of them. Basing the design of communications between OS components on standard network access protocols would ensure larger compatibility and flexible expansion possibilities for the system.

Unfortunately, different manufacturers tend to handle files and data transfers their own way. For increasing compatibility between different systems a general end-to-end protocol, providing users with a set of file access and manipulation facilities, should be implemented on all host computers in a network to ensure efficient host-to-host communication at a level above packet-switching. Since there exists already a set of standards for the lower levels of communications, it is highly desirable that a standard file access protocol be developed in sufficient detail for future implementation.

6. REFERENCES

1. P. Brinch Hansen: Operating System Principles, Prentice-Hall, 1973.
2. The COMTRE Corp., A.P. Sayers, Edit.: Operating Systems Survey, Auerbach, 1971.
3. P. Brinch Hansen: The Programming Language Concurrent PASCAL, IEEE Trans. on Software Engin. 1,2, June 1975.
4. R.S. Fabry: Dynamic Verification of Operating System Decisions, Comm. ACM, vol. 16, no. 11, pp. 659-668, Nov. 1973.
5. W.A. Wulf et al.: HYDRA - The Kernel of a Multiprocessor Operating System, Comm. ACM, vol. 17, no. 6, pp. 337-345, June 1974.
6. D.J. Farber, K.C. Larson: The Structure of a Distributed Computing System - Software, Proc. Symp. Computer - Comm., Networks and Teletraffic, pp. 539-545, Brooklyn, April 1972.
7. E.G. Manning, R.W. Peebles: A Homogeneous Network for Data Sharing: Communications, CCNG Rep. E-12, Univ. of Waterloo, May 1974.
8. H. Aiso et al.: A Minicomputer Complex - KOCOS, Proc. 4th Data Comm. Symp., ACM-IEEE, pp. 5-7 to 12, Oct. 1975.
9. Control Data 6000 Series Computer Systems Reference Manual, CDC Publ. No. 60100000.
10. Control Data Corp. NOS/BE1 Reference Manual, CDC Publ. No. 60493800.
11. D.K. Branstad: Encryption Protection in Computer Data Communications, Proc. 4th Data Comm. Symp., ACM-IEEE, pp. 8-1 to 7, Oct. 1975.
12. C.S. Carr, S.D. Crocker, V.G. Cerf: Host-Host Communication Protocol in the ARPA Network, Proc. Spring Joint Comp. Conf., pp. 589-597, 1970.
13. V. Cerf et al.: Proposal for an International End-to-End Protocol, INWG Gen. Note # 96, IFIP W.G. 6.1, July 1975.
14. H. Zimmermann: The CYCLADES End-to-End Protocol, Proc. 4th Data Comm. Symp., ACM-IEEE, pp. 7-21 to 26, Oct. 1975.
15. H. Zimmermann: Proposal for a Virtual Terminal Protocol, Reseau CYCLADES, IRIA, Jan. 1976.
16. P. Schicker, A. Duenki: Virtual Terminal Definition and Protocol, Comp. Comm. Review, ACM, vol. 6, no. 4, Oct. 1976.
17. P. Schicker, A. Duenki, W. Baechi: Bulk Transfer Function (Proposal), INWG Protocol 31, European Informatics Network, EIN/ZHR/75/20, Sept. 1975.