FIRMWARE / HARDWARE SUPPORT FOR OPERATING SYSTEMS: PRINCIPLES AND SELECTED HISTORY

by

Gary H. Sockut

Center For Research in Computing Technology

Aiken Computation Laboratory

Harvard University

Cambridge, Massachusetts 02138

ABSTRACT

Firmware/hardware support for operating systems is described briefly, and proposed criteria for determining which operating system functions are the best candidates for firmware/hardware implementation are listed. A selected history of the area is presented in three sections: past and current research in support for virtual machines and two sections on past and current research in support for non - virtual machine operating system functions.

This work was supported in part by the Advanced Research Projects Agency under contract F19628-74-C-0083.

INTRODUCTION

Firmware (microprograms) and hardware have traditionally been used to interpret an instruction set (or a high level language) and have left operating system functions to be implemented principally in software. During the past 8 years there have been proposals and implementations in which firmware and/or hardware have been used to support operating systems. This paper discusses principles and examples of such support.

An operating system could be defined as that which manages "resources such as processors, main storage, secondary storage, I/O devices, and files" [Mad74, p. 1]. Sometimes it is difficult (and perhaps inappropriate) to distinguish operating system functions from other functions such as run-time support for a language. For example, an assembly language user of IBM's OS/360 [IBM72] would regard dynamic storage allocation as an operating system function, while an ECL [ECL74] programmer would regard it as part of the language system. Operating system functions will not be defined further in this paper except by example.

Microprogramming will not be defined in this paper; Rosin [Ros73] discusses various definitions.

CRITERIA FOR FIRMWARE / HARDWARE IMPLEMENTATION

Firmware/hardware implementation has three potential advantages over software implementation: elimination of bottlenecks via increased computational speed (efficiency), standardization for software compatibility, and being at a "low level in the system architecture". Accordingly, several criteria are proposed here for determining which operating system functions are the best candidates for firmware/hardware implementation in order to realize the above advantages -- those which:

1) are computation bound rather than I/O bound (such as system monitoring, paging algorithms, storage allocation and deallocation, data structure manipulation, data packing and unpacking for input/output, and interpretation of input/output commands by a classical virtual machine monitor). Firmware/hardware implementation eliminates memory references due to fetching software instructions, but it does not eliminate references that are necessary to access data, unless those data are stored in control store or other fast memory (e.g. for a small working set, such as short tables).
2) are used frequently and use a non-trivial amount of processor resources (such as storage allocation and deallocation, process dispatching, and interrupt handling for fast devices). Hopefully firmware/hardware implementation would significantly reduce the amount of time required to execute such functions.
3) "should be" at a low level (such as security). A software program might be able to bypass software-implemented security mechanisms more easily than it could bypass hardware/firmware-implemented security mechanisms.
4) are non-interruptible (such as synchronization primitives and low-level interrupt handlers).
5) "smooth" the architecture to reduce software costs, complexity, and errors (such as synchronization and communication mechanisms).

PAST AND CURRENT RESEARCH

Almost any hardware/firmware function (such as memory protection) could be said to assist the operating system (or perhaps hinder it), but this paper concentrates on proposals and implementations in which firmware or hardware perform operating system

functions which are typically implemented in software. The emphasis is on firmware. Some of these projects (and others not listed here) also provide high level language support, but operating system support is stressed here. The functions that have been supported or suggested for support by firmware or hardware in the projects described below cover a range of common operating system functions, including virtual machine support, security, process scheduling, semaphores, memory allocation, virtual memory management, data structure manipulation and searching, input/output control and interrupt interpretation, and subroutine linkage and stack storage.

The projects described below are divided into three sections: past and current research in virtual machine support and two sections on past and current research in non - virtual machine operating system support. Projects are listed in chronological order (we believe) within each section. In general, a project's success is not evaluated here. The amount of space devoted to a project does not necessarily represent its significance but instead reflects this author's biases and the amount of relevant information available to him.

Virtual Machine Support

Goldberg has proposed a hardware virtualizer [Gol72; Gol73] as a hardware or firmware mechanism for implementing recursive virtualization via a formal virtual machine map which maps virtual resource names to real (or the next lower level of virtual) resource names. A virtual machine identifier register identifies the mappings for all levels of virtualization, which are composed by the hardware or firmware to define the currently executing virtual machine. To create a higher level of virtualization, a program executing on the real (or a virtual) machine appends a level of mapping to the identifier register. All such lower mappings are logically invisible to programs executing on a virtual machine, which operates as if it were a real machine. A software-visible process map (such as the so called "extended machine" supported by a conventional operating system) can be applied to a highest-level virtual machine. The maps are mutually independent. Goldberg leaves the specific map implementations undefined, since the maps are quite general, but as an example he suggests segmentation for the process map and paging for the virtual machine map. Some difficulties associated with current software virtual machine monitors are eliminated. Examples of such problems on the IBM 360/67 are the requirement of clearing the associative memory after modifying page tables (in supporting recursive virtualization) and the prohibition of modifying a channel program. Also, the hardware virtualizer enables almost every virtual machine operation to be executed efficiently. No description of an actual implementation has been published.

IBM has developed a firmware-implemented Virtual Machine Assist feature [Tal74] to enhance performance of the VM/370 virtual machine system. VM Assist maintains a virtual Program Status Word and other status information for each virtual machine and enables a program executing on a virtual machine to issue supervisor calls, manipulate its Program Status Word and protection keys, and support a higher level of virtual addressing, all without the overhead of trapping to the software virtual machine monitor as would be done at an installation without the VM Assist feature. Statistics indicate that VM Assist does enhance performance; one impressive example is for a particular job stream running under OS/VS1 under VM/370: use of VM Assist resulted in a 65% reduction in total elapsed time [Hor74]. Tallman [Tal75] reports that a principle used in the design of VM Assist was to implement only those functions which are important to performance, after measuring VM/370 overhead. One design constraint was that the new firmware-software interface had to conform closely to the old one (i.e., without VM Assist), in order to minimize changes to the VM/370 software. Another constraint was that the new software had to be able to execute using the old firmware, for software testing and for using the standard firmware in an emergency. VM Assist is not as general as the hardware virtualizer, and it still requires software implementation of such features as virtual input/output. It is significant in that a manufacturer is offering firmware support for virtualization as a regular commercial product.

Project Beta at the University of Southwestern Louisiana [Shr75] is investigating virtualization. Since the project covers a more general range of operating system functions, it is described in the section on current research in non - virtual machine operating system support.

A group at UCLA [Pop75] has modified the hardware/firmware of a DEC PDP-11/45 to enable implementation of and enhance performance of a virtual machine monitor. A PDP-11/45 normally includes ten sensitive instructions which do not result in a trap when executed in a non-privileged mode and which thus prevent virtualization. The UCLA PDP-11/45 CPU has been modified to trap in such cases, so that a virtual machine monitor executing in a privileged mode can simulate these instructions. In addition, a performance improvement unit has been designed as a peripheral device. This unit will interpret most of a virtual machine's references to its upper 4K of memory, which is the mechanism by which a PDP-11 performs I/O and status modification. To implement a virtual machine monitor without this unit, each such reference would have to trap to the virtual machine monitor software, which would then simulate it.

Hitachi Ltd. [Ohm75] is modifying the firmware of its M3 computer, which is compatible with the IBM 370/168, to enhance performance of the VM/370 virtual machine system. The firmware simulates certain privileged instructions, and reflects supervisor calls and certain program interrupts to a virtual machine, without trapping to the virtual machine monitor software. Thus far the modifications have been only for Hitachi internal use.

Past Research (other than virtual machines)

The Burroughs B5700/B6700 series [Org73] provides high-level language support and operating system support. The machine language is postfix. The hardware and firmware provide stack storage, implement block and procedure entrances and exits, and maintain environment records. When a hardware interrupt occurs, relevant data is passed to a software procedure. There are hardware/firmware facilities for tasking, communication, and synchronization. Descriptors assist in data sharing and in segmentation.

The Honeywell Model 8200 [Hat68] features hardware-controlled horizontal multiprogramming whereby single instructions from each of up to 8 programs (plus 1 master program) are executed in round robin sequence. Execution of one instruction is overlapped with fetching the next program's next instruction. The master program can block execution of other programs. Horizontal multiprogramming is claimed to be appropriate for some mixes of I/O-bound programs that make frequent use of fast peripheral devices, since input/output can be overlapped well with processing.

Proposals have been made concerning which functions are most appropriate for microprogrammed implementation. Rosen [Ros68] advocates top-down design of computer systems and suggests interrupt handling, memory allocation, job management, compilation, and debugging aids as suitable functions for firmware implementation.

Werkheiser [Wer70] divides operating system functions into three levels: miniprimitives (such as data structure manipulation and searching and subroutine linkage); midiprimitives (such as interprocess communication and semaphores, memory management, file allocation, and scheduling); and maxiprimitives (such as invocation of assemblers, compilers, loaders, etc.). Werkheiser suggests which primitives for each level are the most appropriate for microprogrammed implementation, based on characteristics such as simplicity and cleanness of interface with other functions, and claims that microprogrammed implementation is appropriate for inter-module communication in a hierarchical operating system. Goals are performance improvement and simplification of software.

The Singer System Ten [Sin73] has a simple round-robin time-slicing supervisor implemented in hardware. Memory partition sizes are fixed at installation time and can be changed later. The goal was apparently to minimize overhead. The Ten is a

decimal machine which was introduced in 1970.

The Venus multiprogramming system [Hub70; Lis72] was implemented at MITRE on a user-microprogrammable minicomputer, the Interdata Model 3. Its purpose was to test the effect of machine architecture on the complexity of software -- system programmers should be able to define the operating system as simply as possible. The operating system is structured hierarchically to provide clarity and reduce errors. Liskov judges Venus' performance to be satisfactory, considering the limitations of the slow hardware; 5 or 6 concurrent users are supported [Lis72]. Venus was constrained by the amount of control store available (2000 microinstructions). The Venus firmware implements segmentation and demand paging, and it invokes a software pager to handle page faults. Processes can share segments, but no protection is provided. The firmware also implements process dispatching, semaphores and waiting queues, and call and return instructions with stack storage to allow shared reentrant procedures. A firmware-implemented input/output channel relieves the software of real-time constraints and gives the illusion of simultaneous input/output processing. I/O completion is signalled using semaphores. Liskov believes that perhaps the only major mistake in the design of Venus was the failure to include measurement mechanisms in the firmware. Such a criticism could also be made of many software operating systems. No further work has been done on Venus beyond that described in the article.

SYMBOL [Ric71] provides hardware implementation of dynamic memory allocation and reclamation, virtual memory management, data structure manipulation, time-sharing supervision, and translation of a high-level language to postfix, which is the machine language. Some functions are implemented via hardware algorithms driven by software parameters, but perhaps firmware rather than hardware implementation would have allowed greater flexibility. The goal of the SYMBOL project was to demonstrate that a high-level language and a large part of a time-sharing operating system could be implemented in hardware to improve performance over that of a software implementation. SYMBOL was produced by Fairchild and is used at Iowa State University. For economic reasons, Fairchild decided not to market SYMBOL.

The unimplemented ISPL system [Bal73] was designed at Rand to integrate a machine, compiler, and operating system via microprogrammed implementation. The machine language is postfix compiled from a PL/I-like source language. The firmware implements process scheduling, memory allocation, and dynamic address translation for segmentation. Processes can share segments. There is a port mechanism for uniform communication with files, devices, and processes. No further work has been done on the ISPL system beyond the planning described in the article, mainly due to unavailability of the planned host machine, which was to be a Standard Computer Corporation MLP-900.

The firmware of the Brown University Graphics System's Meta 4A general-purpose processor [Ana73] implements manipulation and searching of linked lists, tables, character strings, and stacks. It implements subroutine linkage with stack storage for reentrant procedures. The firmware acknowledges input/output interrupts. It also supports extended instructions whereby any instruction whose operation code is illegal is parsed, its operation code, referenced addresses, and operands are stored in main memory, and the processor traps to the operating system, which can simulate any desired instruction. This provides an environment for testing an instruction before committing it to firmware. The hierarchical operating system [Sto73] which is implemented on the processor interprets these extended instructions for such functions as input/output and process synchronization. Goals in the design of the Meta 4A architecture were to reduce memory requirements and execution times and to make assembly language programming convenient, due to the initial unavailability of a compiler. Benchmark programs [Ana73] indicate that the first two goals were met, and the authors' experience indicates that the third goal was met. The Meta 4A is implemented on one of two interconnected Meta 4 processors. It is being used for research in computer graphics and operating systems (see the section on current research). The Meta 4A demonstrates that a user can create an enhanced classical instruction set (IBM 360-like in this case) with some operating system support at moderate microprogramming cost.

Burkhardt and Randel [Bur73] suggest that microprogramming is most useful in improving performance of a hierarchically organized operating system, and that firmware should implement parts of the nucleus directly and should support tasks at higher levels. They suggest implementation of memory protection management, interrupt handling, process dispatching, and process coordination and semaphores, and support for other tasks such as memory allocation and device control via data structure manipulation and search facilities.

One workshop at the Monterey Symposium on the High Cost of Software held discussions on software-related advances in computer hardware [Gag73]. They recommended research into new computer architectures which would simplify programming and debugging and enhance portability. Their recommendations included research into high-level, application-specific machine primitives, descriptor architectures, virtualizable architectures, distributed processing, observability of data structure semantics, and system definition languages.

MCP II [Bel75] is an operating system which is implemented on the Burroughs B1700 [Wil72]. A microprogrammed kernel performs time critical functions such as interrupt handling; implements scheduling, I/O processing, and virtual memory support; and provides communication between various interpreted high-level language programs and the operating system. MCP II is significant in that a manufacturer is offering firmware support for operating system functions in a regular commercial product.

The Honeywell Series 60 Level 64 [Atk74; Atk75] is a medium scale computer with some features found in large scale time-sharing utilities. Firmware and hardware were used as a medium for efficient implementation of security and other time-sharing support features, including process structure and dispatching, segment access protection and sharing, stack storage for reentrant procedures and parameter-passing, list manipulation, interrupt interpretation, and synchronization via semaphores. I/O completion is signalled using semaphores. Two data base management functions (hashing and descriptor-based data field conversion) are implemented in hardware/firmware [Bac75]. The Series 60 Level 64 is an example of how declining hardware costs have led to sophisticated features on a medium scale computer. As with Burroughs' MCP II, a manufacturer is offering firmware support for operating system functions in a regular commercial product.

Current Research (other than virtual machines)

Since 1971, the MEMBERS project [Bro74] at Queen Mary College, University of London, has been developing a system for real-time applications. It includes microprogrammed interpretation of a high-level language, process synchronization, virtual memory management, and instrumentation for the operating system and for applications. It is implemented on an Interdata Model 4 with writable control store.

Since 1973, a group at Brown University has been investigating movement of operating system functions among firmware and levels of software within a hierarchy to improve performance and tailorability for applications [Van75]. They plan to determine the advantages of mobility and to determine necessary support facilities and techniques for the design of operating system functions to allow such mobility. The optimal firmware/software division at any time may depend upon the application, load, etc. Movement may involve modification of data structure definitions, which may be a problem for those which are shared among operating system modules; an operating system design must be flexible enough to accommodate mobility. An interactive monitoring system [Sto75] which drives a graphics display has been constructed to analyze programs static structure and dynamic execution behavior (module interactions and processor utilization). It is used to examine the behavior of the Brown University Graphics System's software operating system [Sto73] and application programs. This measurement should assist in the development of tools with which a user or the system can determine which functions should be moved into low levels in order to best improve performance. A probabilistic model will then be

constructed and will be used to predict performance benefits resulting from changing levels of functions. A limited implementation of software-firmware movement is planned, and a more extensive implementation of movement among levels of software is in progress. A possible additional goal is automation of the choice process.

A group at Bell Laboratories [Bau75] is investigating the microprogramming of operating system functions to enhance performance. They have microprogrammed portions of two routines (storage allocation and release) of IBM's TSS operating system, which is implemented on the IBM 360 model 67. Since IBM does not assist user microprogramming, this group produced the required read-only control store themselves. They had no microcode simulator and thus debugged using the actual machine. A small performance improvement was measured. The project demonstrated that user microprogramming of a complex processor without support from the manufacturer is possible but expensive. Further work in firmware support for operating system functions is planned.

Von Puttkamer [Von75] describes the design of a hardware memory allocator which uses the buddy system allocation algorithm [Kno65]. Shift registers are used to maintain the required binary tree. The goal was to increase allocation speed. When the article was written, an implementation for an Interdata 7/32 was in progress.

Project Beta at the University of Southwestern Louisiana [Shr75] includes research into defining firmware-suitable operating system primitives for multiprogrammed systems. Goals are generality and flexibility via modular resource management and delayed binding of resource control. In this manner, hopefully applications will not be too constrained by low-level design decisions. Early (firmware) binding of primitives should enable efficient implementation of delayed binding of higher-level functions. Related work includes system performance measurement by firmware (with less interference than by software) and error diagnosis by firmware (with a goal of fault tolerance). These goals are to be integrated with virtualization. Project Beta has available three types of processors: MATHILDA [Kor75], a Microdata 3200, and a Honeywell 68/80 MULTICS system.

A group at the University of Ottawa [Per75] has divided operating system functions into three levels similar to Werkheiser's: low (such as data structure manipulation and searching, dynamic address translation, and subroutine linkage); medium (such as process communication, system table maintenance, memory allocation, and file opening); and high (such as invocation of loading, compiling, and linking). They advocate microprogramming of operating system functions for greater speed than software and greater flexibility than hardware. They plan to microprogram an operating system nucleus on a Microdata 1600 computer. Their goals are improvements in performance and reliability.

A group at Carnegie-Mellon University [Ful75] is investigating microprogrammed support for the HYDRA operating system [Wul73], one of whose goals is to provide a small kernel of operating system primitives. This group is using PDP-11/40s with writable control store in a network with other PDP-11s. They are identifying and microprogramming time-consuming operating system functions (such as management of relocation registers) whose performance would be improved by firmware implementation. The network is homogeneous in that a process cannot determine whether the processor upon which it is executing implements such functions in firmware or in software.

CONCLUSIONS

Some of the projects described above use firmware or hardware to implement sophisticated architectural features such as virtualization, security, and run-time support for high-level language execution. Many of the projects use firmware/hardware implementation only to improve computational speed by moving a number of time-consuming functions from software into firmware or hardware. Such movement can enhance performance, but there is nothing magic about the firmware-software boundary. We believe that research in which firmware/hardware

implementation is considered within the context of total system architecture, as advocated, for example, by Rosin [Ros73], will have the most relevance to future architectures. Of the current research projects, those at Brown University and at the University of Southwestern Louisiana appear to be particularly promising in this regard.

ACKNOWLEDGEMENT

The author would like to thank George H. Mealy, Robert P. Goldberg, Richard H. Eckhouse, Jr., Andries Van Dam, Robert F. Rosin, and John E. Stockenberg for reviewing an earlier draft of this paper.

REFERENCES

[Ana73] Anagnostopoulos, P., M. Michel, G. Sockut, G. Stabler, and A. Van Dam, "Computer Architecture and Instruction Set Design", National Computer Conference, 1973.

[Atk74] Atkinson, T., "Architecture of Series 60/Level 64", Honeywell Computer Journal, vol. 8, no. 2, 1974.

[Atk75] Atkinson, T., U. Gagliardi, G. Raviola, and H. Schwenk, Jr., "Modern Central Processor Architecture", Proceedings of the IEEE, June, 1975.

[Bac75] Bachman, C., "Trends in Database Management - 1975", National Computer Conference, 1975.

[Bal73] Balzer, R., "An Overview of the ISPL Computer System Design", Communications of the ACM, February, 1973.

[Bau75] Bauer, S., "Bell Labs Microcode for the IBM 360/67", Eighth Annual Workshop on Microprogramming, ACM SIGMICRO, 1975.

[Bel75] Belgard, R., private communication, April, 1975.

[Bro74] Broadbent, J., and G. Coulouris, "MEMBERS - a Microprogrammed Experimental Machine with a Basic Executive for Real-time Systems", ACM SIGPLAN-SIGMICRO Interface Meeting, ACM SIGPLAN Notices, August, 1974.

[Bur73] Burkhardt, W., and R. Randel, "Design of Operating Systems With Micro-Programmed Implementation", National Technical Information Service report PB 224 484, September, 1973.

[ECL74] "ECL Programmer's Manual", Center for Research in Computing Technology report 23-74, Harvard University, December, 1974.

[Ful75] Fuller, S., private communication, March, 1975.

[Gag73] Gagliardi, U., "Report of Workshop 4: Software-Related Advances in Computer Hardware", in Goldberg, J. (editor), "Proceedings of a Symposium on the High Cost of Software", published by Stanford Research Institute, 1973.

[Gol72] Goldberg, R., "Architectural Principles for Virtual Computer Systems", Ph.D. thesis, Division of Engineering and Applied Physics, Harvard University, 1972.

[Gol73] Goldberg, R., "Architecture of Virtual Machines", ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, 1973; and National Computer Conference, 1973.

[Hat68] Hatch, T., and J. Geyer, "Hardware/Software Interaction on the Honeywell Model 8200", Fall Joint Computer Conference, 1968.

[Hor74] Horton, F., D. Wagler, and P. Tallman, "Virtual Machine Assist: Performance and Architecture", IBM New England Programming Center technical report 75.0006, April, 1974.

[Hub70] Huberman, B., "Principles of Operation of the Venus Microprogram", National Technical Information Service report AD 709 717, July, 1970.

[IBM72] "IBM System/360 Operating System - Introduction", IBM publication GC28-6534-4, 1972.

[Kno65] Knowlton, K., "A Fast Storage Allocator", Communications of the ACM, October, 1965.

[Kor75] Kornerup, P., and B. Shriver, "An Overview of the MATHILDA System", ACM SIGMICRO Newsletter, January, 1975.

[Lis72] Liskov, B., "The Design of the Venus Operating System", Communications of the ACM, March, 1972.

[Mad74] Madnick, S., and J. Donovan, Operating Systems, McGraw-Hill, 1974.

[Ohm75] Ohmachi, K., private communication, August, 1975.

[Org73] Organick, E., Computer System Organization - The B5700/B6700 Series, Academic Press, 1973.

[Per75] Perez, A., D. Banerji, and J. Raymond, "Microprogrammed Operating Systems: A Design and Implementation Proposal", Computer Science Department, University of Ottawa, 1975.

[Pop75] Popek, G., "The PDP-11 Virtual Machine Architecture: A Case Study", Fifth Symposium on Operating Systems Principles, ACM SIGOPS, 1975.

[Ric71] Rice, R., and W. Smith, "SYMBOL - A Major Departure From Classic Software Dominated Von Neumann Computing Systems", Spring Joint Computer Conference, 1971.

[Ros68] Rosen, S., "Hardware Design Reflecting Software Requirements", Fall Joint Computer Conference, 1968.

[Ros73] Rosin, R., "The Significance of Microprogramming", International Computing Symposium, Davos, Switzerland, September, 1973; and ACM SIGMICRO Newsletter, January and July, 1974.

[Shr75] Shriver, B., T. Lewis, and J. Anderson, "Integrated Research Projects in Virtual Computer Systems: Project Beta", Computer Science Department report, University of Southwestern Louisiana, January, 1975.

[Sin73] Singer Business Machines, "System [Ten] Summary Manual", 1973.

[Sto73] Stockenberg, J., P. Anagnostopoulos, R. Johnson, R. Munck, G. Stabler, and A. Van Dam, "Operating System Design Considerations for Microprogrammed Mini-Computer Satellite Systems", National Computer Conference, 1973.

[Sto75] Stockenberg, J., and A. Van Dam, "STRUCT Programming Analysis System", First National Conference on Software Engineering, 1975; published in IEEE Transactions on Software Engineering, December, 1975.

[Tal74] Tallman, P., R. Denson, T. Gilbert, J. Nichols, and D. Stucki, "Virtual Machine Assist Feature Architecture Description", IBM Poughkeepsie Laboratory technical report 00.2506, January, 1974.

[Tal75] Tallman, P., "Virtual Machine Assist Feature Microcode Implementation", in Microprogramming and Systems Architecture, Infotech State of the Art Report 23, Infotech International Ltd., 1975.

[Van75] Van Dam, A., "Performance Improvement Through Function Migration in a Distributed Computing System", proposal to the National Science Foundation, Division of Applied Mathematics, Brown University, September, 1975.

[Von75] Von Puttkamer, E., "A Simple Hardware Buddy System Memory Allocator", IEEE Transactions on Computers, October, 1975.

[Wer70] Werkheiser, A., "Microprogrammed Operating Systems", Third Annual Workshop on Microprogramming, ACM SIGMICRO, 1970.

[Wil72] Wilner, W., "Design of the Burroughs B1700", Fall Joint Computer Conference, 1972.

[Wul73] Wulf, W., E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "HYDRA: The Kernel of a Multiprocessor Operating System", Department of Computer Science report, Carnegie-Mellon University, June, 1973; and Communications of the ACM, June, 1974.