Toward a Methodology of Protection Implementation (In TENEX: Memo 1)

Chuck Wall

February 4, 1974

Technical Memo

TM,74-4

1. Introduction

This memo is a very brief introduction to a proposed method of protection implementation in programmed systems. In particular, this discussion will be limited to an examination of the TENEX JOB, FORK and FILE objects at their design level. In general, this memo takes the point of view that protection implementation in new or existing systems should be approached on a sound methodological basis. The ad-hoc transferring of "protection mechanisms" from one system to another on the theory that "what is good for the goose is also good for the gander" is precisely what should be avoided.

We do not argue that the methodology proposed is in any way the only approach or the best approach, but only that Ad-Hoc procedures be avoided. There has been a considerable amount of research into security and protection in programmed system. The methodology proposed in this memo is based primarily on the research into 'Models of Protection" by Jones [J73] and the Hydra system of Wolf, et. al [W73]. We believe that Jones' Model is general enough to cover a wide range of protection policies.

In this memo we will attempt some tentative definitions of various entities and provide some informal interpretations in the hope of provoking further discussion. We then introduce Jones' 'Model" and attempt to evaluate TENEX protection from the viewpoint of that 'Model". Finally we describe the proposed methodology at an introductory level.

2. Background Information

The Hydra system [W73] was designed to facilitate "the creation of highly secure systems". Protection is a specific consideration in the Hydra design philosophy:

> "Protection: Flexibility and protection are closely related, but not inversely proportional. We believe that protection is not merely a restrictive device imposed by 'the system' to insure the integrity of user operations, but is a key tool in the proper design of operating systems. It is essential for protection to exist in a uniform manner throughout the system, and not to be applied only to specific entities (e.g., files). The idea of capabilities (in the sense of Dennis [DVH66] is most important in the Hydra design; the kernal provides a protection facility for all entities in the system. This protection includes not only the traditional read, write, execute capabilities, but arbitrary protection conditions whose meaning is determined by higher-level software."

The Hydra designers consider protection in the following framework:

> "Protection is, in our view, a mechanism. A system utilizing that mechanism may be more or less secure depending upon policies governing the use of the mechanism (for example, passwords and the like are policy issues) and upon the reliability of the programs which manipulate the protected entities. Thus the design goal of the Hydra protection mechanism is to provide a set of concepts and facilities on which a system with an arbitrarily high degree of security may be built, but not to inherently provide that security. A particular consequence of this philosophy has been to discard the notion of 'ownership'. While ownership is a useful, perhaps even important, concept for certain 'security' strategies, to include the concept at the most primitive levels would be to exclude the construction of certain other classes of truly secure systems."

This is a reasonable viewpoint for implementation, but in this memo it will be useful to talk about protection policies and protection mechanisms when we refer to protection. By protection policies we mean both the global policies that a systems' protection mechanisms enforce and the individual user policies the protection mechanism enable. The Hydra designers soundly reject "hierarchical system structures" as a possible protection mechanism.

> "Our rejection of hierarchical system structures, and especially ones which employ a single hierarchical relation for all aspects of system interaction, is also, in part, a consequence of the distinction between protection and security. A failure to distinguish these issues coupled with a strict hierarchical structure leads inevitably to a succession of increasingly privileged system components, and ultimately to a "most privileged" one, which gain their privilege exclusively by virtue of their position in the hierarchy. Such structures are inherently wrong, and are at the heart of society's concern with computer security. Technologists like hierarchical structures; they are elegant, but experience from the real world shows they are not viable security structures. The problem, then, which Hydra attempts to face squarely, is to maintain order in a non-hierarchical environment."

This is a very strong viewpoint, but one which should be considered.

Lampson describes an operating system (CAL-TSS[L69b]) that is "built around a very powerful and general protection mechanism." The central protection mechanism in this system is called a "capability" and performs the following two functions:

1. it names an object in the system
2. it establishes a right to do certain things with the object

The system contains a convenient method of grouping "capabilities" together to form classes of "capabilities" that provide "a rather flexible means for authorizing various things to be done without requiring the possession of a large number of individual capabilities."

Freeman [F74] discusses "Protection Mechanisms and Policies in the BCC 500 Operating System." The protection policies governing the mechanisms in this system are based on an "ownership" attribute. The BCC 500's protection mechanisms should provide some significant insights into the pro's and con's of the "ownership" attribute. A more complete evaluation of these mechanisms is forthcoming.

Jones [J73] defined a "Model of Protection" that will be a very useful tool in the examination and comparison of protection abilities of various actual and proposed operating systems. This thesis forms the bases of the methodology proposed in this memo.

3. Protection Defined

We adopt Jones' views on protection that specify:

"A programmed system is a set of algorithms implemented in software and hardware, mapping values input to that system to output values. Protection is the enforcement of rules to maintain order during the mapping activity. To be more precise the mapping activity is described as a set of distinguishable processes each performing a sequence of algorithms defined in the programmed system. The processes share the use of data retaining objects. At any time each process is aware of a set of objects and the specified ways each object can be used. Order is maintained and interactions are made limited and predictable by rules which regulate the acquisition and use of access to objects. The enforcement of these rules is called protection."

and establish the following:

Limits of the protection problem--a View from a Larger Context

"Protection is a part of a larger concern, called security, that encompasses the regulation of activity both internal and external to the programmed system. A secure computer system depends upon

1. controlled access to objects within the programmed system--the topic addressed within this thesis
2. reliable hardware components
3. prevention of threats perpetrated outside the programmed system, for example, monitoring or tapping communication lines, theft of data volumes, or illegal entry into the programmed system through falsified identification.

To isolate protection from the other components of the security problem, we assume that all physical resources used to implement a computer system are completely reliable and free from external tampering. We assume infallible verification of the identity of users requesting service from the system. Since the protection component of security address controlling access to entities within a closed universe assumed to be free from any external influences, it can be studied separately."

Jones considers a "protection system" as being composed "a policy and a set of mechanisms built from hardware and software, the enforce and implement policy." It is useful to extend these ideas slightly to consider having a set of policies (where each policy is independent of every other policy in the sense that they do not conflict) and that the mechanisms may possibly allow users to implement individualized protection policies that provide for more or less protection than that which is provided by the basic system policies. For example, if it is a system policy that user directories are only accessible to the directory owner and the list of users the owner specifies, the mechanisms should provide for the ability to make a particular directory public without attaching a list of all users to some access directory.

4. Definitions

It is convenient to be able to utter words that 'mean exactly what we want them to mean, nothing more and nothing less'.† We will, of course, be unable to attain this goal, but will offer several definitions and informal remarks in order to clarify our meaning. We hope also that by doing this we can provoke discussions that will lead to better definitions. All of the definitions to be present are motivated by Jones' thesis. We will attempt to improve slightly on a few definitions, but will accept Jones' definitions. In those cases that we modify the definitions we will provide both versions.

Definition: (Jones') Object - refers to software-defined structured encoding of information to which access is to be controlled as well as to physical resources (for example, a block of memory or a device)

(ours) include virtual resources (for example, a system call and other operations that may be implemented in firmware or software)

Remarks: An object is in general anything that can potentially be accessed and includes such things as files, subsystems, system calls, operations, processes, processors, core,...etc.

Definition: (Jones') Environment - a table of rights each expressed as an (object-name, access-name) pair restricted so that an access is named in a right only if it is applicable to the object named in that right.

†An approximate "quote" of a original Lewis Carrol idea.

Remarks: An "object-name" is the set of one or more global names that refer to a specific object. It is assumed that each object is uniquely identifiable. So there may be a many to one mapping from object-names to an object, but not a one to many (two or more) mapping from object-name to object. Clearly devices like tape drives can be referenced by either a mneumonic name or physical device number. The access-name simply specifies the access rights that a particular environment has to a particular object. Naturally an access-name cannot specify execute-access to a text file which is not capable of being executed.

Definition: (Jones') Type - The set of all objects is partitioned into equivalence classes, each called a type

Most people who have been associated with the idea of time sharing have at least an intuitive idea of what a process is about. Nonetheless, it is very difficult to precisely define a process. Lampson gives one viewpoint which is useful [L69a].

> A process is a vehicle for the execution of programs, a logical processor which shares the physical processor with other programs. It consists of a machine state, some resources it expends in doing work, and some additional structure to describe its memory and its access to objects.

Another useful viewpoint is that a process is the smallest object in the system that can be independently scheduled for execution.

5. Introduction to Jones' "Model of Protection"

Jones' thesis defines a very general protection model and a method of comparing protection policies in various operating systems. The model is based on "three rules that are to be bound in order to describe any individual system:

- Enforcement Rule
- Right Transfer Rule
- Environment Binding Rule."

A comparison method called a "suitability measure" is based on " the need-to-know principle: any any time a process has in its execution environment only those rights required to perform the current task."

We briefly introduce the model and discuss some of the important points we will need later.

5.1 The Enforcement Rule

- The Enforcement Rule specifies that a process can exercise only those rights which are in its current operating environment.

In this specification the word "current" indicates that a process exists in different environments at different times and operating simply means that the process is active. Jones introduces the idea of "enforcement of protection" very nicely as follows:

> "A process sequentially exercises rights as specified by a program. In general it is possible that the program directs an executing process to violate protection, i.e., to exercise a right that is not in the execution environment. Therefore either static compile time protection checks or dynamic run time protection checks must be made to detect attempted protection violations. If a check reveals that a protection violation will occur, some action to avoid the violation must be taken. For example, in the dynamic case, execution by the offending process could be aborted. In the static case, the attempt to use a variable that is not available could cause automatic declaration of a new variable with the same local name.

> A process distinguishes an object that is to be referenced by using a name for it. All names that can be uttered by the process comprise its current name space. Because a process can only reference those objects for which its environment contains a right, we can assume that there exists one name space associated with each execution environment. The name interpretation function f_E maps the name space associated with environment E to the set of all objects.
>
> $$f_E: \{\text{names}\} \xrightarrow{\text{interpretation path}} \{\text{objects}\}$$
>
> Because name interpretation necessarily accompanies each exercise of a right, correlating the performance of name interpretation and protection checking will guarantee that all requisite checks are made in support of the Enforcement Rule. We assume that every object is uniquely identifiable, both in our discussion and in actual systems. For the purposes of this thesis each object is identified by its unique global name. (To facilitate readability of the following prose, no strict distinction is made between the use of the terms 'object' and 'global name of an object'. The term 'object' is used for either of the two.)
>
> The name interpretation function can be as simple as the identity function, if the name space of a process is the set of global names. (More complex name interpretation functions will appear in later examples.) The name interpretation function itself can be used as a form of protection. This can be achieved by letting the range of f_E be a proper subset of all objects. The result is that a process in E cannot access any object which is not in the range of f_E, for it cannot name such an object."

Jones limits the model to "the study of environments which are directly represented, i.e., there exists an encoding of environments independent of other types of variables in the system. The motivation for considering only direct representation is so that access to environments can be controlled as for any other type." Jones specifies the following three protection checking mechanisms that provide protection enforcement at:

- Execution Site
- Object Site
- Access Site

3.1.1 Execution Site Protection Enforcement

Jones states that execution site protection enforcement is the following:

> "A first extension of the name interpretation function incorporates protection checking at the beginning of the interpretation path--the site of execution, where local names are generated. For each environment there exists a capability list representation (Dennis & Van Horn [DVH66]). Each right in an environment E is encoded as a capability of the form (global object name X, access name A)."

The name interpretation function f_E is extended to include protection checking and is called F_E. Jones then states that:

> "Encoding protection data in an environment structure at the site of execution has the following advantages:
>
> 1. It provides a name space (names are selectors into the capability list) that includes names only for objects accessible from within that environment.
> 2. It isolates within a single data structure all protection information pertaining to execution in each environment. (This structure need only be available when execution in the environment is in progress.)
> 3. Protection checking can be performed without leaving the execution site. By definition, F_E prevents all violations of the Enforcement Rule."

We note the following disadvantages:

1. Once an environment obtains a capability to access an object it must be guaranteed that the rights with respect to that object for that environment do not change during the time the environment has the capability or that any change in a right is immediately reflected in the capability that represents it locally.
2. It is not a viable method in instances where object-names are dynamically generated since there is no way to know in advance what the name will be. For example, if the process requests the user to supply the name of a file to be accessed it is very doubtful that the capability to access that file will already exist at the execution site.

There are however many instances when this type of protection checking mechanism will afford efficient protected access. We do note however that implementation of this particular type of protection mechanism in the general sense of Hydra's [W73] "Local Name Space" (LNS) requires a considerable amount of machinery since an LNS is uniquely associated with an invocation of a procedure. But it is indeed possible to consider particular cases where this type of protection mechanism could be usefully implemented on existing systems.

5.1.2 Object Site Protection Enforcement

This form of protection simply means that protection checking is done at the end of the name interpretation path. It is usually implemented as some form of access list.

5.1.3 Access Site Protection Enforcement

Here protection is associated with some point in the path from object-name to access-name. An example, given by Jones, of this type of protection is that of type checking as provided by Algol 68 compilers.

Jones also considers the idea of "interleaving of name interpretation and protection checking" and static verses dynamic checking.

5.2 Right Transfer Rule

"The Right Transfer Rule of the protection model specifies that every protection system must define a policy to regulate the movement of rights into and out of environments. In any particular system both the right transfer policy and its implementation must be bound. (This contrasts with the Enforcement Rule which is fixed for every protection system that the model covers, although implementations of it vary, for example with the representation of environments.)" Jones goes on to examine various right transfer policies and to specify a set of primitive operations and later "demonstrates that these operations do form a set of building blocks sufficient to construct a variety of protection policies."

5.3 Environment Binding Rule

"The Environment Binding Rule of our model requires that each protection system specify how a process can cross between environments." In this rule Jones considers the actions and circumstances involved in environment crossing and introduces a new primitive operation (AMPLIFY) "for cases in which the right transfer primitives are insufficient to describe modification of a parameter right as it enters the (target) environment." Jones notes in the summary that "because parameter passage operations are short, well-defined and often executed, and therefore are candidates for implementation in microcode or hardware."

5.4 Suitability Measure

Finally Jones discusses a method of comparing various protection systems. Jones defines an accuracy measure and a suitability factor based on a need-to-know principle. "This principle can be stated as: A process performing a task in some execution environment should have access to only the objects required in the performance of that task."

6. TENEX Objects

This memo is limited in that it deals only with the TENEX Job, Fork and File objects at a design level rather than the implementation level. This is quite reasonable at this stage of the development of this series of memos and more in-depth studies will be persued as the methodology is developed and refined.

6.1 Job

In TENEX a Job is associated with the following:

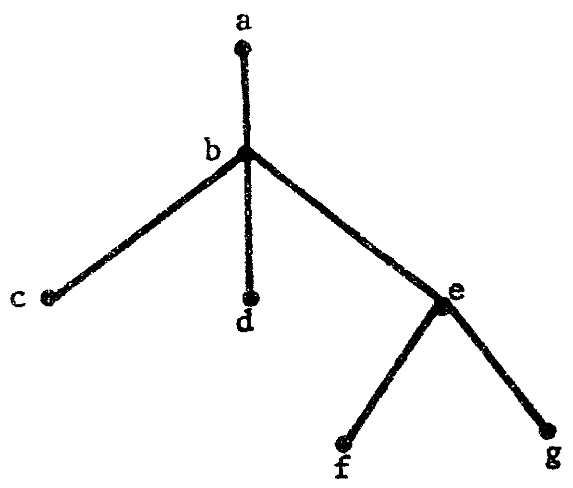
1. The user who initiated the job
2. An account number
3. The user's file directory
4. A hierarchy of running and/or suspended processes
5. A virtual memory

Information about the job is contained in a Job Storage Block. This block is composed of one or more pages and is referenced by the monitor and Exec. It does not appear in any user process' virtual memory.

The user obtains a job from the job pool at login. The only information maintained from the last time the user performed a login is that which has been preserved in files. At logout all files are closed, all forks are deleted and the job is returned to the job pool.

6:2 Forks

Each job is permitted to have multiple simultaneously runnable processes called forks. A typical fork structure is displayed in Fig. 6.2 below.



Parallel Pointer	Inferior Pointer
Superior Pointer	fork handle

Fork Table Format

Notes: b is a subsidiary fork of a
c, d, e are subsidiary forks of b and are parallel forks
f, g are subsidiary forks of e and are parallel forks

Figure 6.2 Fork Structure

The top level fork in all jobs is the exec. It is possible for a fork to create inferior (subsidiary) forks but not parallel or superior forks in the structure. A fork is the smallest object in TENEX that is individually scheduled for execution. Communication among forks is accomplished by:

- sharing memory
- termination, initiation, or suspension of a subsidiary fork
- pseudo (software simulated) interrupts

A fork may be assigned capabilities, but these in general are related to either the job or the user capabilities. The capabilities a fork may be passed are as follows:

Job Capabilities

process can activate terminal key PC for pseudo interrupt

process can examine monitor tables with GETAB

process can map the running monitor

process can execute protected log functions process can map privileged pages of files

Capabilities (amplified) which can be given to an inferior whether or not the superior itself has them.

process can do map operations on superior

unprocessed PSI's cause freeze instead of termination

User Capabilities

user is a wheel

user has operator privileges

user can see system confidential information

user can HALT TENEX

A fork, when created is assigned an area of storage called Process Storage Block (PSB). The create fork monitor call will assign the PSB and add the fork to the fork hierarchy in the JSB.

6.3 Files

In general the TENEX file system provides a repository for information in the conventional sense and also gives the user access to devices not usually associated with files.

In this memo we are concerned with the protection mechanisms available in TENEX. To this end we refer the reader to Appendix 1 which contains a description of that protection that has been copied from the TENEX EXEC manual.

7. Relating the Model to TENEX Objects

In terms of the model, all the protection mechanisms I have seen occur at the access site. The passing of "capabilities" to forks can be analyzed using the models rights transfer rule. Environment Binding appears to occur primarily at the job level. Monitor calls are implemented as procedures and appear to be good candidates for the extension of protection along the lines suggested by Jones. That is monitor procedures may implement environment binding at a low level.

8. TENEX Compared with the Systems Jones Compared

Jones gives the following partial ordering of four systems:

$$0 < S(OS/MVT) < S(MULTICS) < S(CAL\text{-}TSS) < S(HYDRA) < 1$$

where "the notation S(Z) is used to distinguish the suitability factor for the system called Z." We note that this partial ordering is based in part on the level in which environment binding occurs. To this end then the suitability scale of these systems with respect to this level is as follows:

$$0 < S(USER) < S(PROCESS) < S(PROCESS\text{-}PROCEDURE) < S(PROCESS\text{-}PROCEDURE\ CALL) < 1$$

Since TENEX is a job oriented system which affords more protection than a USER system but not more than a complete process system we can locate TENEX as follows:

$$0 < S(OS/MVT) < S(TENEX) < S(MULTICS) < S(CAL\text{-}TSS) < S(HYDRA) < 1$$

9. The Proposed Methodology

The proposed methodology is simply to consider each (TENEX at this time) system as being composed of the following three levels with respect to the protection function:

- Policy
- Logical Design
- Implementation

Then to analyze the policy level for logical soundness and consistency by using Jones' "Model of Protection". Next determine and be able to specify the mapping from policy to logical design and from logical design to implementation.

This approach naturally assumes a reasonably complete documentation effort at each of the three levels. Since TENEX documentation presently available is not organized from the viewpoint of protection, our present interest is in providing this documentation. Once this effort is completed a detailed analysis of TENEX at each of the three levels can proceed, logical flaws or minor bugs can be noted and TENEX can be more accurately categorized with respect to protection and more realistically compared with other systems.

Given a detailed analysis (of TENEX) alternative policies can be explored and the mapping process logically followed to the implementation level, giving some idea of the effort involved in a particular policy change. This method enables a structured, stagewise approach to the protection implementation problem and affords the opportunity to determine in advance the effect of various policy changes.

APPENDIX I

Description of TENEX File Protection taken from the EXEC Manual:

2. File Protection

In general, one can apply any TENEX file command to any disk file, no matter who it belongs to as long as one can specify the descriptors that make up its file designator. Furthermore, even when one's knowledge of these descriptors is incomplete, TENEX can help fill in the gaps through its recognition and default value features. Thus, by its nature, TENEX provides a very general file sharing capability. This leads to an equally general need for some way of protecting files from unauthorized access.

To this end, TENEX provides a file protection mechanism, exactly balancing its file sharing capability. File protection is controlled by a six digit octal number assigned to each file at the time of its creation. This number breaks down into three fields, identical in format, each of which can be regarded as containing two octal digits or six binary bits. The bits in any one of these fields control the following aspects of file protection.

```
FILE     PROTECTION     FIELD
B0   B1   B2    B3    B4   B5
```

Bit	Protection Aspect Controlled
B0	Read contents of file
B1	Write onto file
B2	Execute program stored in file
B3	Append to file
B4	Access per page table
B5	Not used

Setting a bit 1 permits the indicated action; setting it Ø denies the action. Read allows information to be extracted from a file. Write permits new information to be written onto a file, replacing part or all of the original contents. Execute allows a file that has been read into core memory to be executed as a program. Any file can be made executable in this sense; what happens when one tries to start execution, of course, depends on what the file actually contained. Append allows new information to be added to the end of a file. "Per page table" access means that read, write, and execute access will be specified by the file itself for each individual page. The remaining bit is unused and can take on either value with no effect.

Taking these six bits as a two digit octal number, some common values are: 77, which permits full access; 52, which protects a file from modification, but permits other functions; and ØØ which denies everything.

As mentioned above, a full, six digit file protection number contains three of these protection fields, arranged as follows:

<u>FILE PROTECTION NUMBER</u>
<u>self</u> <u>group</u> <u>others</u>

Working from left to right the three fields control access to the file by successively larger groups of users. Self protection governs one's own access to a file. Limiting one's own access permits one to protect valuable information from inadvertent destruction, or to protect a file from modification by a possibly faulty program.

3. Directory protection

In addition to allowing detailed specification of the access to file contents, TENEX also allows the contents of directories to be protected in a similar way. The format of the directory protection word is composed of three 6-bit fields, one field for each of "self", "group", and "others" similar to the file protection word. The bits have meaning as follows:

Bit	Protection Aspect Controlled
BØ	If off, completely prevents use of the directory in any way
B1	Files may be opened subject to file protection
B2	Owner-like functions may be performed (including CONNECT) without password
B3	Files may be added to the directory
B4	Not used
B5	Not used

The directory protection word may be changed by contacting the TENEX Operations Manager.

4. Groups

TENEX provides a mechanism whereby users can form groups for the purpose of file sharing. This is useful, for example, where several users are collaborating on a common programming job and wish to share the files they are creating. The file group mechanism works as follows. Each group is assigned a number. For each user, TENEX records a series of these numbers indicating what groups he belongs to. Likewise, for each file directory, TENEX records numbers indicating which groups have access to the files it contains. Thus a given user can belong to a number of groups. He can make each of his directories available to one or more groups, which may or may not coincide with the groups he belongs to. The assignment of group numbers to users and to file directories is done by the TENEX system operators.

The second field in a file protection number governs the powers granted when the file is accessed through the group mechanism. If a group member designates a file in a directory available to his group but belonging to another user, it is this protection field that controls the kind of access granted him.
The third protection field governs the access granted to users approaching the file from outside the group mechanism, i.e., all "other" users. In a typical situation, the three protection fields might specify successively more stringent protection. For example a user might grant full access to himself, read, execute, and append access to group members, and append only access to others, yielding ;P775404 as protection descriptor.

If a user omits a protection number when creating a disk file, TENEX will assign a default value, presently 777754. This grants full access to the user and group members, read, execute and append access to others. The Default File Protecion word may be changed by contacting the TENEX Operations Manager.

BIBLIOGRAPHY

[B73] Bolt, Beranek and Newman, Inc., TENEX JSYS Manual, Cambridge, Mass., Sept. 1973.

[DVH66] Dennis, J. B. and Van Horn, E. C., "Programming Semantics for Multiprogrammed Computations," CACM 9,3, March 1966, 341-346.

[F74] Freeman, Jack, Memorandum on the Protection Mechanisms and Policies in the BCC 500 Operating System, TM.74-1, ALOHA SYSTEM (Task II), University of Hawaii, Jan. 1974.

[J73] Jones, Anita Katherine, Protection in Programmed Systems, Ph. D. thesis, Carnegie-Mellon University, Dept. of Computer Science, 1973.

[L69a] Lampson, B. W., CAL-TSS Internals Manual, Computer Center, University of California, Berkeley, November 1969.

[L69b] Lampson, B. W., Dynamic Protection Structures, Proc. AFIPS 1969 FJCC, Vol. 35, AFIPS Press, Montvale, N. J., 27-38.

[O72] Organick, E. I., The Multics Systems: An Examination of its Structure, MIT Press, 1972.

[P73] Price, W. R., Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems, Ph. D. thesis, Carnegie-Mellon University, 1973.

[W73] Wolf, W. A., et. al., HYDRA: The Kernal of a Multiprocessor Operating System, Carnegie-Mellon University, Computer Science Department report, June 1973.