

DRI File Copy

ESD ACCESSION LIST

DRI Call No. 84146

Copy No. 1 of 1 cys.

INITIAL STRUCTURED SPECIFICATIONS FOR AN
UNCOMPROMISABLE COMPUTER SECURITY SYSTEM

K. G. Walter
W. F. Ogden, et al
Case-Western Reserve University
Cleveland, Ohio 44106

July 1975

Approved for public release;
distribution unlimited.

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
HANSCOM AIR FORCE BASE, MA 01731



ADA022490

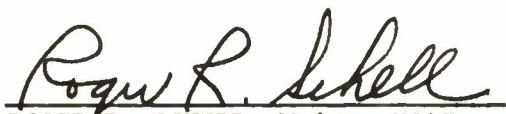
LEGAL NOTICE

When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

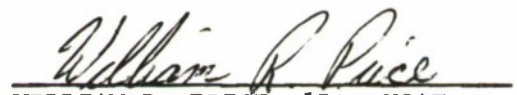
OTHER NOTICES

Do not return this copy. Retain or destroy.

"This technical report has been reviewed and is approved for publication."



ROGER R. SCHELL, Major, USAF
Techniques Engineering Division



WILLIAM R. PRICE, 1Lt, USAF
Techniques Engineering Division

FOR THE COMMANDER



STANLEY D. DERESKA, Colonel, USAF
Chief, Techniques Engineering Division
Information Systems Technology
Applications Office

The specifications of the Security Kernel are developed through a series of successively more complex models which are used to specify the system in increasing detail. The first specification, S_0 , defines in general what it means to say that a system is uncompromisable or that there can be no unauthorized disclosure of information. Subsequent models introduce directory structure, interprocess communication, file and process attributes, and other system functions. Using this approach, we can discuss various design issues related to security in an orderly and straightforward manner.

INITIAL STRUCTURED SPECIFICATIONS FOR
AN UNCOMPROMISABLE COMPUTER SECURITY SYSTEM

By K. G. Walter, W. F. Ogden, J. M. Gilligan, D. D. Schaeffer,
S. I. Schaen, D. G. Shumway

Project: Abstract Model for Computer Security, Contract No.
F19628-73-C-0185, ESD, USAF, Hanscom Field, Bedford, Mass. 0173

Department of Computing and Information Sciences,
Case Western Reserve University, Cleveland, Ohio 44106

Dated: July 1975

ACKNOWLEDGEMENTS

The authors wish to express their thanks to the many people who have provided insight and guidance throughout the course of this project. We are especially indebted to Dr. Franklyn Bradshaw, Dr. William Rounds, Stanley Ames, and Kenneth Biba, former members of the CWRU Computer Security group, who contributed substantially to the results presented in this report. In addition, we are grateful to Major Roger Schell, Lieutenants Paul Karger and William Price of the Air Force Electronic Systems Division, and to James P. Anderson for supplying background information and for their frequent comments and constructive suggestions. We would also like to express thanks to Edmund Burke, Leroy Smith, Richard Rhode, C. Chandersekaran, Jon Millen and Judah Mogilensky of the MITRE Corporation for their directional guidance and critiques of the several interim reports.

PREFACE

This technical report is the final report documenting the results of a two year effort at Case Western Reserve University. This effort produced a mathematical model of security in computer systems. The model mathematically represents the Department of Defense Information Security Program and establishes sufficient criteria for security controls to prohibit the unauthorized disclosure of information contained in computer systems.

The modeling effort provides the technical foundation for the multifaceted Automatic Data Processing (ADP) Security Program sponsored by the Air Force's Electronic Systems Division. The model guided the implementation of security enhancements for a now operational Air Force computer system, the Honeywell Multics system at the Air Force Data Services Center. Use of this system to process information of multiple classifications requires operation of the system in a "non-malicious" environment because the controls were not analytically proven to be totally effective. A separate, on-going phase of the Security Program has the goal of developing a secure general purpose prototype computer system with security controls which are similar to those of the operational system but which have been proven effective a priori. The security of the prototype system will be proven by demonstrating the correspondence of the implementation (through a series of intermediate design representations) to the model.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	ii
PREFACE	iii
1. INTRODUCTION	1
1.1 Motivation	1
1.2 The Multics System	2
1.3 The General Goal.	3
1.4 The Proposed Approach	4
1.5 A Short History of Work in The Area	5
1.6 Structured Specification.	7
1.7 An Outline of the Report	9
2. THE BASIC STRUCTURE, S_0	10
2.1 Introduction	10
2.2 Basic Elements, Functions and Relations	13
2.3 The Air Force Security Lattice.	18
2.4 Need To Know Security Enforcement	21
2.5 The Relationship Between Successive Levels of Structured Specification	23
3. THE S_1 STRUCTURE: FILE AND MAILBOX STRUCTURES	26
3.1 Introduction	26
3.2 File and Mailbox Structures	27
3.3 The Mathematical and Axiomatic Structure of S_1	29
3.4 The Consistency of S_1 and its Relationship to S_0	34
4. FORMALIZATION OF DYNAMIC SECURITY	35
4.1 Introduction	35
4.2 A Formal Description of a Dynamic Secure System	40
4.3 An Informal Description of Some Dynamic Problems.	49
4.4 The Dynamic Security of S_2	57
5. THE SECURITY EVENTS IN S_3	64
5.1 Introduction	64
5.2 Event Descriptions	67
5.3 Proofs Concerning Events	82

TABLE OF CONTENTS (continued)

	Page
6. S_4 - COMMAND AT THE SECURITY PERIMETER	83
6.1 Introduction	83
6.2 Process	84
6.3 The PST	85
6.4 Files	87
6.5 Access Attributes	88
6.6 Characteristic Attributes	90
6.7 Process Attributes	97
6.8 Primitive Operations	99
7. CONCLUSION	126
7.1 Introduction	126
7.2 Summary	127
7.3 Further Work To Be Done	129
7.4 Other Problems	129
7.5 Future Topics	131
REFERENCES	132
APPENDIX A: A General Specification for Information Passing Systems, S_{-1}	134
APPENDIX B: Formal Description of S_1 Consistency and the S_1 - S_0 Relationship	139
APPENDIX C: A Proof of Proposition 4.4.1	145
APPENDIX D: Proofs Concerning Events	150
APPENDIX E: Mappings and Proofs from S_4 to S_3	168
NOTATION	181

1. INTRODUCTION

1.1 Motivation

The Air Force, as well as many other government agencies, have already accumulated a great deal of national security as well as other sensitive information in computer systems throughout the world. It is likely that the use of computerized systems for information storage and retrieval will become even more pervasive in the near future. Protecting this computerized information from unauthorized disclosure is a problem because traditional security measures are not always directly applicable to computer systems. In fact, dealing with this new environment necessitates a reexamination of the purposes of the Department of Defense Information Security Program governing classified information.

Dedicating a separate computer facility to a particular classification of information would solve many security problems in Air Force applications. However, it is more feasible as well as financially preferable to combine a number of small computer installations into a single large installation. Unfortunately, current computer operating systems lack sufficient certifiable correctness of design needed to provide sophisticated security to their users. This lack of provable soundness would even concern small installations where it would be desirable to enforce "need-to-know" restrictions on the dissemination of information.

In addition, there are operational requirements in the military which necessitate the sharing of large computer data bases among many subgroups. This sharing is used to promote coordinated and efficient

operation of a computer system as well as reduce the cost of maintenance and other system overhead.

1.2 The Multics System

Multics is a sophisticated time-sharing system with a large shared file data base. This system was developed at Massachusetts Institute of Technology in cooperation with Bell Labs and General Electric and is currently supported by Honeywell on a 68/80 machine. Multics has evolved with several features which are of interest from a security point of view. For example, the access control lists on the user files certainly appear sufficient for the military's "need-to-know" security requirements.

The 68/80 hardware also provides several protection mechanisms which keep the run-time cost for security checking from being prohibitively high. The access control bits on each entry in the segment table of a process allow selective access restrictions to be enforced on each instruction execution. The eight protection rings of Multics provide nested domains which can be used to separate the security system and more vital portions of the operating system from users as well as the remaining less reliable parts of the operating system.

1.3 The General Goal

The overall objective of the current Air Force project, of which we are a part, is to develop a certifiably secure computer system using a Multics-like machine thereby providing the Air Force with a useful as well as secure system. In the process it is important to develop design techniques which can be used in developing secure operating systems for other machines and other applications. Arriving at a precise definition of Computer Security for military purposes is a particularly significant part of this second goal. The design technique used in this effort should ultimately lead not only to a system which is secure, but also to a system which can be proven secure.

1.4 The Proposed Approach

Because of its enormous size and complexity, it is difficult and tedious to analyze an operating system while considering it as a whole. Fortunately, however, it has proven feasible to isolate the security related portions of an operating system in a comparatively small software module which we call a Security Kernel. In essence, the security kernel will consist of the elements of an operating system needed to verify and monitor the transfer of information around the system. Protection hardware will be used to completely isolate the security kernel from external programs. The implementation of this security kernel will provide the user with a virtual machine which will have somewhat different memory, IO control, and instruction set than those provided by the bare hardware of the 68/80. (E.g. the virtual machine will not appear to have a disk memory, but rather a "directory structured" memory.) This virtual machine is referred to as the Security System.

To facilitate certifying the correctness of the security kernel it must be kept as small as possible. However, all security related aspects of the entire system must be considered in the design of the security kernel. Accordingly, the current Multics operating system must be very carefully dissected to determine the portions which must go inside the security kernel and those which can remain outside in the residual operating system.

Our task then, is to develop a sound set of specifications for the security kernel. These specifications should take into account the security features of the hardware which will be available to the actual implementors of the security kernel.

1.5 A Short History of Work in the Area

In 1969 Butler Lampson [7] presented an abstract framework within which to discuss computer security. He identified and described the concepts of subjects and objects as the active and passive elements in a computer system. His subjects and objects were used as prototypes for our work and correspond roughly to our agents and repositories. The privileges of the subjects to manipulate objects was determined by an access matrix which was maintained by the security system.

Weissman, in a subsequent effort, attempted to apply governmental security restrictions to a computer system in his Adept 50 system [16]. The Adept 50 system was an operating system designed for an IBM 360 model 50 which included some features of governmental security.

In 1971, the Air Force Electronic Systems Division formed a security panel to discuss Air Force needs and responsibilities in the area of computer security. Anderson [2] reported the conclusions of this panel. Two of the recommendations of this panel are of particular significance to the current project. First, the panel concluded that the Air Force should invest in an effort to develop secure operating systems. Secondly, the panel recommended that such an effort should begin by developing sound mathematical models of the desired system. The research reported in this paper represents a portion of this modeling effort.

In 1973, Schell, Downey, and Popek [12] articulated their preliminary views on how the mathematical models of a security system should be formulated. Bell and LaPadula [4] followed up these preliminary views with considerably more elaborate versions of the

proposed mathematical models.

Our efforts on this project began in the spring of 1973. We soon discovered certain necessary security requirements of the system. We realized that a major component of the problem was to develop a language in which to discuss the solution. We found that this language was necessary both to describe the desired system and eventually to carry out the proofs that the implementation was correct. We quickly found that discussing the rationale behind certain design decisions necessitated the use of certain auxiliary concepts which would have no specific realization in the final implementation.

Our first attempts resulted in a rough model corresponding approximately to the present S_2 specification in this paper. In formulating this model, we discovered two design principles which evolved into the acquisition and dissemination axioms of the S_0 specification. We developed S_0 to emphasize the importance of these principles and to formulate a firm foundation for future development. We then developed the S_1 specification to discuss tree structured file systems and to explain our somewhat surprising conclusion that the classification of files must not decrease as you progress away from the root of the tree.

1.6 Structured Specification

One of the more important results of our work has been the development of a design technique which we call structured specification. In brief, this technique consists of systematically developing detailed specifications from an initial general specification by introducing additional system details through a sequence of refinement steps.

This technique has the advantages of allowing system analysis to take place in an environment less cluttered with irrelevant system details, as well as allowing large problems to be broken into smaller, more tractable tasks. In addition, structured specification encourages the discovery of similarities between different parts of the system and of system wide principles.

The structured specification technique initially involves formulating a simple specification for the system (in this paper S_0). Next, a more detailed level of specification, usually having somewhat different sets, functions, and relations, is created. It must then be established that the new specification is simply a refinement of the previous specification. This is accomplished by identifying the old sets, functions and relations within the new specification and then proving that the axioms of the old specification are consequences of the axioms in the new specification. This process of creating and proving levels of specification continues until the newest specification can be implemented in software on the hardware of the available computer.

The axioms of the final level of specification constitute the assertions which must be proved about the implementation in order to

establish that it is "correct". Since each level of specification has been proven to be a refinement of its predecessor, the final implementation will be accurately described by each level of specification (in particular, the first level, S_0).

1.7 An Outline of the Report

The second through fifth chapters of this report have been organized in accordance with the structured specification methodology discussed previously. Chapter two is devoted to the S_0 specification which constitutes our basic definition of an uncompromisable computer system.

Chapter three discusses the S_1 level of specification which introduces the concepts of mailboxes for interprocess communication and directories for file system organization. The S_2 level of specification discussed in the fourth chapter of this report, introduces some of the dynamic aspects of the security system. The S_2 specification abstractly describes the manner in which the access control bits in the Multics segment descriptor tables are to be used to enforce security. This chapter also contains a discussion of the relation between "dynamic" and "static" models of security.

The fifth chapter presents the S_3 specification which gives a considerably more detailed account of the proper handling of the file and process attributes. It includes an extensive list of security system commands which must be developed when developing the security kernel. In the sixth chapter these commands are related to a set of prototype Multics commands similar to those being developed by the MITRE Corporation.

The seventh and final chapter serves as a summary of our work to date and discusses some problems which remain to be solved. In addition, there is a discussion of several issues related to the larger project of which this effort is a part.

2. THE BASIC STRUCTURE, S_0

2.1 Introduction

In this chapter we present S_0 , our most abstract level in the structured specification process. In this specification, the details of the target operating system are abstracted away to enable us to inspect the most basic objects and actions of the system. By looking at these system fundamentals we were able to formulate the necessary axioms to restrict the flow of information and guarantee that there can be no unauthorized disclosure of information. Because of its generality, S_0 has become our definition of mandatory security satisfying the military requirement that it "permits access to information only as defined by the rules governing dispersal of classified information" [13]. S_0 further serves as a guide for the subsequent structured specification refinements.

Following the pioneering work of Lampson [7] and others we present S_0 as an abstract system in which the information store is partitioned into a set of disjoint repositories (objects) and the processing of and transferring of information is accomplished by a set of agents (domains or subjects). The repositories are viewed as passive information storing elements. The information stored in a repository can only be changed by one of the agents, and such modifications can only be detected by other agents through an observation. Since in this structure all transfers are from repository to repository through an agent, it is possible to control information flow by controlling the observation and modification privileges of each of the agents.

Since the repository is the smallest unit of information storage to which observation and modification access is controlled, all information stored in a repository is considered to be of uniform sensitivity. As a measure of this level of sensitivity, a security class is associated with each repository. The set of repositories is subdivided into disjoint subsets of repositories with the same security class; each subset will therefore have the same mandatory access restrictions.

Rather than keep lists specifying which agents can access which set of repositories, we associate a security class with each agent. The security class then becomes the common measure which makes the set of agents and set of repositories comparable and which can be used to determine what observations and modifications are to be permitted.

For motivation of the method of solution and of the terminology, let us consider the Air Force regulations and procedure for preventing compromise of classified information. For this purpose, repositories are documents, and agents are personnel. The security class of a document is its classification, and the security class of an individual is his clearance. An individual may read (observe) a document only if its classification is less than or equal to his clearance. Notice that this is a necessary but not a sufficient condition. It gives a basis for controlling observations, but there must also be a basis for determining which modifications should be allowed. Recall that our objective is to prevent unauthorized disclosure.

Therefore, though not explicitly required by the Air Force regulations information must not be transferred to a repository with a security class lower than that of the source repository. To prevent agents

from transferring information to a repository with insufficient security class, an agent will not be allowed to modify a repository unless its security class is greater than or equal to that of the agent.

In order to discuss this comparison between elements we must define a relation on the set of security classes. Let us consider what properties the relation must have. Certainly an agent with a given security class (clearance) should be allowed by the mandatory security system to observe or modify any repository with the same security class (classification). Then since every security class must be less than or equal to itself, the relation must be reflexive. In order to allow possible information transfer from one repository to another through one or more intermediate repositories, it is necessary that the relation be transitive.

According to Popek [10] in his general treatment of access maps using restriction graphs, the relation on the set of security classes can be a partial ordering. However, antisymmetry is not essential for the basic theorem of this chapter and it is sufficient to assume that the relation on the set of security classes is a pre-ordering.

Conceptually a repository is any information storing element in the system. More practically, however, a repository can be looked upon as any object in the system which can be modified in such a way that this modification can be subsequently observed. The proposed S_0 specification is intended to be sufficiently general to include all possible information channels. We believe that to leave large classes of potential information channels to be haphazardly discovered and dealt with by the system implementer is to circumvent the very idea of design through structured specification.

2.2 Basic Elements, Functions, and Relations

We now introduce some notation and formalize the ideas discussed in the preceding section.

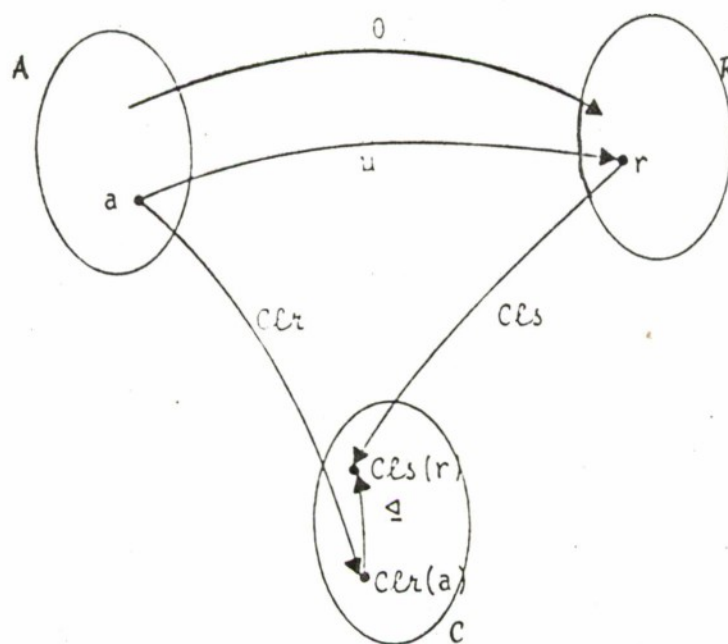
The mathematical structure for mandatory security in an information processing system is an 8-tuple:

$$S_0 = (R, A, C, \theta, \mu, \preceq, Cls, Clr)$$

where

- | | |
|--------------------------------|--|
| R | is a set of repositories. |
| A | is a set of agents. |
| C | is a set of security classes. |
| $\theta \subseteq A \times R$ | is the "observe" relation. ($\underline{a} \theta \underline{r}$ means that agent $\underline{a}$ can observe the information stored in repository $\underline{r}$.) |
| $\mu \subseteq A \times R$ | is the "modify" relation. ($\underline{a} \mu \underline{r}$ means that agent $\underline{a}$ can modify the information stored in repository $\underline{r}$.) |
| $\preceq \subseteq C \times C$ | is a pre-ordering of the set of security classes. |
| $Cls: R \rightarrow C$ | is the "classification" function which associates a security class with each repository. (Informally $Cls(\underline{r})$ will be referred to as the classification of repository $\underline{r}$.) |
| $Clr: A \rightarrow C$ | is the "clearance" function which associates a security class with each agent. (Here again $Clr(\underline{a})$ will be referred to as the clearance of agent $\underline{a}$.) |

The structure S_0 can be illustrated by the following picture:



The mathematical structure S_0 has four axioms. The first two are technical axioms which simply state explicitly that the relation $\leq$ is a pre-ordering of the set C of security classes.

A0.1 (Reflexivity): For all $c \in C$, $c \leq c$.

A0.2 (Transitivity): For all $c, d, e \in C$, $c \leq d$ and $d \leq e$ implies $c \leq e$.

The second two axioms state the rules governing the acquisition and dissemination of information by agents.

A0.3 (Acquisition): For all $a \in A$ and $r \in R$, $a \theta r$ implies $Cls(r) \leq Clr(a)$. (That is, if agent a can observe repository r , then the clearance of a must be greater than or equal to the classification of r .)

A0.4 (Dissemination): For all $a \in A$ and $r \in R$, $a \mu r$ implies $Clr(a) \leq Cls(r)$. (That is, if an agent a can modify repository r , then the clearance of a is less than or equal to the classification of r . Agent a can modify only those repositories with equal or higher security class.)

Using these four axioms, we now prove that in S_0 no information can ever be transferred to a repository in which it can be observed by an agent that does not have sufficient clearance to observe the source repository.

In preparation for the formal statement and proof of the basic security theorem about S_0 , we make the following definitions. Define the "transfer" relation $\tau \subseteq R \times R$ by $r \tau s$ if and only if there is an agent a such that $a \theta r$ and $a \mu s$, where r and s are repositories. Thus $r \tau s$ means that there is an agent which can trans-

fer information from repository $\underline{r}$ to repository $\underline{s}$. It is actually the transitive, reflexive closure τ^* of τ which is needed in the theorem, since $\underline{r} \tau^* \underline{s}$ means that information can eventually be transferred from $\underline{r}$ to $\underline{s}$. Specifically $\underline{r} \tau^* \underline{s}$ means a sequence of agents and repositories exists through which information could be transferred from $\underline{r}$ to $\underline{s}$. Accordingly, we say an information transfer path exists from $\underline{r}$ to $\underline{s}$ when the relation $\underline{r} \tau^* \underline{s}$ holds.

THEOREM

TO.1: If there is an information transfer path from repository $\underline{r}$ to repository $\underline{s}$ in S_0 , then $Cls(\underline{r}) \leq Cls(\underline{s})$,

PROOF: By definition, if there is an information path from repository $\underline{r}$ to repository $\underline{s}$, then $\underline{r} \tau^* \underline{s}$. We first establish that $\underline{r} \tau \underline{s}$ implies $Cls(\underline{r}) \leq Cls(\underline{s})$. For if $\underline{r} \tau \underline{s}$, then there is an agent $\underline{a}$ such that $\underline{a} \theta \underline{r}$ and $\underline{a} \mu \underline{s}$. By axioms A0.3 and A0.4, $Cls(\underline{r}) \leq Clr(\underline{a})$ and $Clr(\underline{a}) \leq Cls(\underline{s})$. By transitivity of $\leq$, $Cls(\underline{r}) \leq Cls(\underline{s})$.

Now define a new relation $\lambda \subseteq R \times R$ by $\underline{r} \lambda \underline{s}$ if and only if $Cls(\underline{r}) \leq Cls(\underline{s})$. (That is, $\underline{r} \lambda \underline{s}$ means the security class of $\underline{r}$ is "less than or equal" to the security class of $\underline{s}$.) Notice that λ is a reflexive and transitive relation. For any $\underline{r} \in R$, axiom A0.1 states that $Cls(\underline{r}) \leq Cls(\underline{r})$, and so $\underline{r} \lambda \underline{r}$ (i.e., λ is reflexive) by the definition of the relation λ . If $\underline{r}, \underline{s}, \underline{t} \in R$, such that $\underline{r} \lambda \underline{s}$ and $\underline{s} \lambda \underline{t}$, then, by the definition of λ , $Cls(\underline{r}) \leq Cls(\underline{s})$ and $Cls(\underline{s}) \leq Cls(\underline{t})$.

By axiom A0.2, $\text{Cls}(\underline{r}) \subseteq \text{Cls}(\underline{t})$, and, again by definition of λ , $\underline{r} \lambda \underline{t}$ (i.e., λ is transitive).

The relation λ contains the relation τ , since $\underline{r} \tau \underline{s}$ implies $\text{Cls}(\underline{r}) \subseteq \text{Cls}(\underline{s})$, which implies $\underline{r} \lambda \underline{s}$. Recall that by definition of the transitive closure, the relation τ^* is the minimal transitive, reflexive relation containing the relation τ . It follows that the relation τ^* is contained in the relation λ . This proves the theorem, since, for $\underline{r}, \underline{s} \in R$, $\underline{r} \tau^* \underline{s}$ implies $\underline{r} \lambda \underline{s}$, which implies $\text{Cls}(\underline{r}) \subseteq \text{Cls}(\underline{s})$.

2.3 The Air Force Security Lattice

In this section we hope to demonstrate the applicability of S_0 to a system which enforces the Air Force clearance/classification and compartmentalization restrictions. We do this by showing that these two schemes (Clearance, Compartments) can be combined to give a single set of security classes. Air Force "need-to-know" restrictions have not been included because, as will be discussed later, strict mandatory enforcement of need-to-know gives a system of limited usefulness, and it does not model the actual military use of need-to-know.

Let us now write out the details of the security lattice which describes the Air Force classification system. Let Sen be the set of sensitivity levels (i.e., unclassified, confidential, secret, etc.). Sen is a lattice because it is linearly ordered.

Let Cmp be the set of compartments of subject matter (China, nuclear, etc.). Generally the information stored in a given repository may be included in more than one compartment; hence, the component of a security class concerned with compartmentalization will actually be a subset of compartments to which the information belongs. Although all possible subsets of Cmp may not be needed in practice, our formal treatment will use the entire power set $P(Cmp)$ of Cmp . $P(Cmp)$ is a lattice naturally ordered by set inclusion. The two lattices Sen and $P(Cmp)$ can be combined to form the product $Sen \times P(Cmp)$ which is itself a lattice [8, p. 489] with order relation defined as follows:

for $(c,D),(e,F) \in Sen \times P(Cmp)$, $(c,D) \trianglelefteq (e,F)$ if and only if $c \leq e$ in Sen and $D \subseteq F$ in $P(Cmp)$. Thus an agent may observe a repository only if the classification level of the repository is less than or equal to the clearance level of the agent and the set of compartments associated with the repository is a subset of the set of compartments associated with the agent.

Since the set $C_0 = Sen \times P(Cmp)$ is a lattice, under the ordering $\trianglelefteq$ defined above, axioms A0.1 and A0.2 will be satisfied. Being a lattice, C_0 has stronger properties than required for mandatory security as defined by S_0 . While not necessary, those additional properties may be useful in practice. For example, given any two classes in C_0 , there is a minimal class which is greater than or equal to either one. Also, C_0 has a greatest and a least element.

The basic theorem T0.1 then states that, in a system which uses the Air Force security lattice C_0 to restrict the observe and modify relations according to axioms A0.3 and A0.4, there can be no transfer of information from a repository with one sensitivity and set of compartments to a repository with lower sensitivity or smaller set of compartments. Also, since the only access to repositories is through agents, there can be no unauthorized disclosure of information.

The modeled system is quite similar to the real world except for axiom A0.4 which states that an agent may not modify a repository with lower security class. Under that restriction, if agents are acting on behalf of system users, a user who has, say, secret clearance could not send any information to an uncleared user through the system. This is necessary since the system cannot interpret which information is clas-

sified and which is not.

To realistically apply the basic theorem to the Air Force security procedure, we might suggest that a person be allowed to operate as an agent with any clearance up to and including his actual clearance. This places the responsibility on the user to decide at which level he should operate. In making the decision to operate at a reduced clearance, the user relinquishes the right to observe any material classified higher than his "working level." In this way axiom A0.4 can be satisfied and the result of the theorem ensured.

2.4 Need To Know Security Enforcement

The Air Force further ensures the privacy and integrity of information by requiring that to access information one must not only have proper clearance but must also have established a "need-to-know" for the information. The fact that Jones is cleared to see material in a given security class does not mean he can read a document with that classification written by Smith, unless he has been extended the proper need-to-know authorization.

Initially we attempted to include the need-to-know security scheme in the basic security model by incorporating it into a lattice structure. However, this security structure proved much too rigid to be of any use, except perhaps in the special case of a small environment. As an alternative to this lattice type structure we present a less strict need-to-know security system in which the responsibility for protecting the contents of repositories is left to the individual users. We will refer to this system as the Discretionary Security System.

This system involves attaching to each repository a list of users who are allowed to observe the contents of that repository. There is a major disadvantage to this system: if we let user Smith observe one of our repositories, we cannot be certain that he will not pass the contents along to user Jones. This is similar however to the real world situation.

An advantage of the discretionary security system is that we only have to check whether one user is on a need-to-know list rather than having to check whether one list of users is contained in another list of users. Checking for individual membership is usually much faster than checking for containment.

There is another security problem which the discretionary security system should confront, and that is the problem of sabotage. We could construct a strict "need-to-modify" security system to deal with this problem, but it would be just as cumbersome as the strict "need-to-know" system. The proposed discretionary security system will deal with this problem by attaching to each repository a list of users who are allowed to modify the contents of that repository.

Since the discretionary security is not as strict about controlling access to repositories as the mandatory security is, we will provide the user with additional mechanisms for controlling his agents. These will take the form of agent privileges which the user may selectively revoke.

2.5 The Relationship Between Successive Levels of Structured Specification

S_0 has both intuitive appeal and formal simplicity. The major design requirement that the system permit no compromise of information has been achieved and controlled sharing of multi-level information has been accounted for. All that is needed is to look at an actual computing system, show that the specifications which completely define its operation can be formulated into a precise mathematical structure, demonstrate that these formal specifications are consistent, and that the four axioms of S_0 hold under the interpretation.

This procedure is trivial for a trivial system, however it is not at all clear how it can be applied directly to a complex operating system such as Multics. The formalism of axiomatic systems has been used to concisely formulate the intuitively obvious in precise terms, but the power and effectiveness of the method have not been demonstrated. The concept of S_0 is thought to be sufficiently general to describe a broad class of information security problems including applications which use the Air Force sensitivity-category classification lattice as their set of security classes and to permit software implementation in a segmented architecture. However it gives little direction towards actual implementation.

The technique of structured specification will now be illustrated by giving a second level structure which will satisfy the security requirements in S_0 plus further design requirements to describe a file system structured as a tree of arbitrary depth and a mechanism for "inter-agent" communication which does not require accessing a shared file. These additional restrictions make the design more implementa-

tion specific. We note further that such additional structures are not required for prevention of compromise of security but are to analyze certain design decisions at an abstract level of discussion before formal specifications for implementation are attempted.

At each new level of specification, additional restrictions will be imposed. Difficulties in formulating a consistent axiomatic system which validly interprets the preceding structure and satisfies the new restrictions can indicate two different problems. Either the newly added restrictions are not compatible with a secure design or the previous level was not appropriate. The first case demands a change in design decision at the new level; the second calls for a reformulation of the preceding level. The technique is thus one of successive, possibly reiterated, refinements of design requirements each followed by a consistent formal specification which validly interprets the previous structure and which incorporates the new design concepts. The process is complete when at some level the mapping between formal symbols and actual system objects is a trivial identification and all desired restrictions have been incorporated. At this point, formal proof of the security of the final structure is a relatively straightforward result of composing the several one-to-one correspondences between levels to yield a single correspondence between it and the S_0 specification. Proving the entire system will also require establishing that all the axioms of the final level do in fact hold in the implementation of the system.

As noted before, because of its generality and intuitive obviousness it is not expected that it will be necessary to reformulate S_0 .

Beyond that it is hoped that reiteration can be minimized by informally weighing the expected consequences of design decisions before their formalization. The possibility of the need to backup and modify a previous level should not be construed to be a weakness of the method in the sense that the formal system at any level is at sometime not secure. The machinery of the axiomatic method precludes this. The need to reiterate indicates a weakness in the designer in that he is unable to comprehend a complex operating system and to make all the correct decisions in one step. Structured specification then is a tool which allows the designer to proceed a step at a time toward his ultimate goal. Each step is guided by previous steps, by previous mistaken steps, and by general intuitive notions about the final goal. Since the designer does not know precisely where he wants to go he must accept the probability of making a few wrong steps along the way. Incorrect steps may provide further insight about the system design. Also, since no code will have been written nor logic built, the cost of a mistaken step will be minimal.

3. THE S_1 STRUCTURE: FILE AND MAILBOX STRUCTURES

3.1 Introduction

The S_0 structure defined information security with an abstract, presumably intuitive, description of a secure system. This chapter presents a structure, S_1 , relating the S_0 definition of security to computer system design. The development of two basic features of modern computer systems, file systems and processes, will be initiated. The repository concept of S_0 is refined by subdividing the set of repositories into two distinct subsets. One subset is explicitly structured as a tree of files. The other subset of repositories provides a mechanism for communication between agents.

3.2 File and Mailbox Structures

A subset of S_0 repositories is identified in S_1 as files (called segments in Multics). To reflect the Multics directory naming scheme of addressing segments we organize these files into a tree structure. To store or retrieve information in a given file, the file system must be searched from the root until the file has been found; therefore, restrictions must be imposed on the relationships between files' locations in the tree and their classifications. These restrictions are developed in the axiomatic description of section 3.3.

Agents can communicate through the modification and subsequent observation of files. This method of communication however would be subject to all the restrictions imposed on the file structure. It is desirable to provide a less cluttered mechanism devoted to communication between agents; therefore, we introduce a non-file communication path between agents. The mechanism modeled will be sufficient to implement those features which Multics provides.

Objects such as Multics' event channels and semaphores could be named globally and referenced through the file system. Instead we designate as mailboxes a subset of the set of S_0 repositories. Mailboxes are not files and are not restricted by file considerations and usages. Being repositories, mailboxes do have classifications, and agents can in the S_0 sense observe and modify them. Two new relations, send and receive, provide agents the ability to communicate through mailboxes. Sending to and receiving from mailboxes will be restricted with the necessary assertions governing the use of them. The mechanism thus designed should be general enough to support more refined mechanisms.

Since an agent's passing information to another agent of higher classification does not compromise security, agents may send to mailboxes of equal or higher classification but not to lower classifications. Receiving from a mailbox may constitute both an observe and a modify in the S_0 sense depending upon the implementation of the mailbox mechanism. A mailbox mechanism is likely to be implemented as a message queue; an agent's receiving a message would involve observing the message and removing it from the front of the queue. This removal can be observed by another agent if the mailbox has classification equal to or lower than the agent's clearance. (See [14, appendix B] for further explanation.) By the S_0 observe restrictions, receiving must be from a mailbox of equal or lower classification than the agent receiving; however, the modify restriction disallows receiving from a lower classification. Agents therefore may receive only from mailboxes having classification equal to their clearance. Communication between agents of unequal clearances can occur since sending to higher classifications is allowed. As necessitated by S_0 an agent may not communicate information down to an agent of lower classification.

The restrictions on the send and receive relations are applicable whether the implemented operations are strictly simple synchronizing signals or whether the operations permit the transmission of a message with the signal. These restrictions are stated explicitly in the axioms which follow.

In more refined structures communication between agents, interagent communication, will be refined. When processes are introduced, mailboxes will provide the means for interprocess communication.

3.3 The Mathematical and Axiomatic Structure of S_1

The structure for mandatory security in an information processing system with hierarchically structured files and inter-agent communication is the following:

$$S_1 = (F, M, A, C, \rho_F, \sigma_F, \rho_M, \sigma_M, \underline{a}, \delta, Cls, Clr)$$

where

F is a tree of files (directories and segments)

M is a set of mailboxes

A is a set of agents

C is a set of security classes

$\rho_F \subseteq A \times F$ is the "retrieve information" relation.

(a ρ_F f means that agent a can retrieve information from file f.)

$\sigma_F \subseteq A \times F$ is the "store information" relation. (a σ_F f means that agent a can store information in file f.)

$\rho_M \subseteq A \times M$ is the "receive" relation. (a ρ_M m means that agent a can receive information through mailbox m.)

$\sigma_M \subseteq A \times M$ is the "send" relation. (a σ_M m means that agent a can send information to mailbox m.)

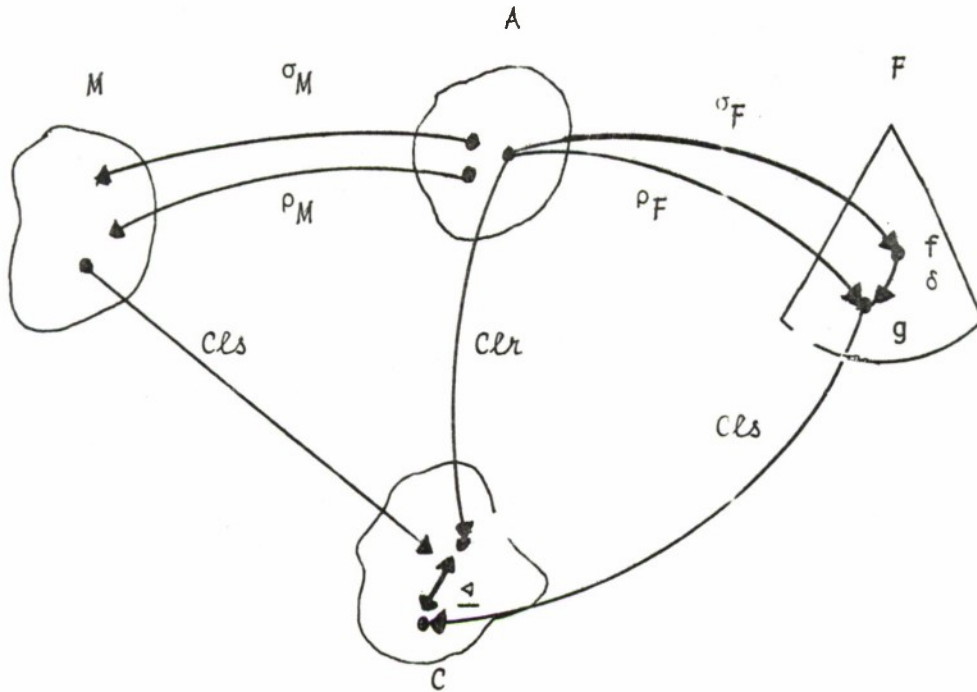
$\underline{a} \subseteq C \times C$ is a pre-ordering of the set of security classes.

$\delta \subseteq F \times F$ is the "dominate" relation on the set of files.
(It defines the "tree" structure on the files.)

$Cls: F \cup M \rightarrow C$ is the "classification" function for files and mailboxes.

$Clr: A \rightarrow C$ is the "clearance" function for agents.

M_1 can be illustrated by the following diagram:



We now list the axioms for S_1 .

- A1.1: For all $c \in C$, $c \leq c$ ($\leq$ is reflexive).
- A1.2: For all $c, d, e \in C$, $c \leq d$ and $d \leq e$ implies $c \leq e$ ($\leq$ is transitive).

The next four axioms impose restrictions on the store, retrieve, send, and receive relations in order that S_1 may validly interpret S_0 . The retrieve and receive relations correspond to observation and the store and send relations to modification.

- A1.3: For all $a \in A$ and $f \in F$, $a \rho_F f$ implies $cls(f) \leq clr(a)$.
(An agent can only "retrieve" information from a file with equal or lower classification.)
- A1.4: For all $a \in A$ and $m \in M$, $a \rho_M m$ implies $cls(m) = clr(a)$.

(An agent can only "receive" information through a mailbox with classification equal to its own clearance.)

A1.5: For all $a \in A$ and $f \in F$, $a \sigma_F f$ implies $Clr(a) \leq Cls(f)$.
(An agent can only "store" information in a file with equal or greater classification.)

A1.6: For all $a \in A$ and $m \in M$, $a \sigma_M m$ implies $Clr(a) \leq Cls(m)$.
(An agent can only "send" information through a mailbox with equal or greater classification.)

The tree structure on the set of files is asserted by the following:

A1.7: For all $f \in F$, $f \delta f$ (δ is reflexive).

A1.8: For all $f, g \in F$, $f \delta g$ and $g \delta f$ implies $f = g$.
(δ is antisymmetric).

A1.9: For all $f, g, h \in F$, $f \delta g$ and $g \delta h$ implies $f \delta h$.
(δ is transitive).

A1.10: For all $f, g, h \in F$, $g \delta f$ and $h \delta f$ implies $g \delta h$ or $h \delta g$ (δ has the "tree" property).

The remaining axioms formalize the use of tree structure on the files as a directory.

A1.11: For all $a \in A$, and $f, g \in F$, $a \rho_F g$ and $f \delta g$ implies $a \rho_F f$. (In order to retrieve information from a file, an agent must be able to retrieve from (i.e. search) every file which dominates it. This axiom recognizes certain implicit information paths which result from the nature of the directory system. To illustrate the difficulty, suppose that an agent $\underline{a}$ deletes a directory $\underline{f}$ which

dominates a file g ; then another agent a can tell that the file g is no longer observable. This constitutes a potential communication path between a' and a . In effect, a can make a limited observation of directory f by determining whether file g is still observable. (See Lampson [6] for a discussion of similar problems.)

A1.12: For all $a \in A$, and $f, g \in F$, if $a \sigma_F g$ and $f \delta g$ and $f \neq g$ then $a \rho_F f$. (In order to store into a file, an agent must be able to retrieve from or search every file which strictly dominates it. Note that it is possible for an agent to store into a file which he is not allowed to read; thus, "writing-up" to a higher classification is allowed. Any implementation which permits such "writing-up" must be careful to avoid introducing any implicit information paths. For example, if an error occurs in writing to a file of higher classification, no error message may be given; otherwise, this would provide a method of communication from higher to lower classification.)

Proposition 3.0: For any agent in A and any files f and g in F , if $a \sigma_F g$ and $f \delta g$, then $Cls(f) \leq Cls(g)$. (A file which can be modified by some agent must have a classification which is greater than or equal to the classification of any directory which eventually contains it.)

The proof of Proposition 3.0 follows from previous axioms:

Axiom A1.5 says that, if $a \sigma_F g$, then $Clr(a) \leq Cls(g)$. Now note that A1.12 asserts that, if $a \sigma_F g$, then $a \rho_F f$. By axiom A1.3 $Cls(f) \leq Clr(a)$. Using the transitivity of $\leq$ (Ax2), $Cls(f) \leq Clr(a) \leq Cls(g)$ implies $Cls(f) \leq Cls(g)$ as desired.

The implications of Proposition 3 are clearer if we state it in contrapositive form.

Corollary 1: For any agent a in A and files f and g in F , if $f \delta g$ and $Cls(f) \not\leq Cls(g)$, then not $a \sigma_F g$.
(No agent can modify a file g which is contained in a directory whose classification is not less than or equal to the classification of the file g .)

This corollary motivates us to introduce another axiom in order not to have a file system which is cluttered with useless, unmodifiable files.

A1.13 For any files f and g in F , if $f \delta g$, then $Cls(f) \leq Cls(g)$. (Every directory has an equal or lower classification than any file it eventually contains. The root of a directory tree must have the lowest classification existing in the tree; furthermore, the sons of any node in the tree can be classified no lower than that node.)

3.4 The Consistency of S_1 and its Relationship to S_0

An S_1 structure has been described and the formal axioms of the structure have been stated. It is necessary to formalize the relationship between the S_1 structure and the S_0 structure; S_1 must be proved to be a refinement of S_0 . The formal statements and detailed proofs of this assertion are in appendix B of this report.

All objects in the S_1 structure must be shown to be instances of objects in the more abstract S_0 structure. A proposition is stated and proved that a one-to-one mapping from S_1 to S_0 exists so that the relations of S_0 are preserved. To complete the proof of refinement, it is formally shown that every valid interpretation of S_1 (S_1 type structure) is a valid interpretation of S_0 and consequently any theorem of S_0 is true in S_1 also.

Although it is not necessary in proving that S_1 is indeed a refinement of S_0 , a proposition states that S_1 is consistent, that is, that no self contradictions exist within it. The proposition is proved by constructing a specific S_1 type structure which satisfies the axioms. If S_1 was not consistent no such example could be constructed. Although S_1 need not be consistent to be an interpretation of S_0 , this proposition is proved first since an inconsistent structure would be of no value in the structured specification process.

4. FORMALIZATION OF DYNAMIC SECURITY

4.1 Introduction

As we progress from the very abstract description of controlled information sharing to a more complex structure which can serve as a formal specification for the implemented system, we must make certain design decisions. For example, the file concept introduced in S_1 was an abstraction of an information bearing object. Beyond the facts that each file had an associated classification and that files were organized hierarchically nothing was assumed about other attributes of the files. A further simplification in S_1 is the lack of any consideration of time-variant aspects of an actual system. At this point it is time to look at available technology and make some decisions about feasible ways of implementing the relationships asserted in S_1 .

It was demonstrated in chapter 2 that the intended application of S_0 theory to the Air Force classification/compartmentalization scheme was possible. If the resultant lattice of security classes is stored directly as a bit vector (value) rather than in some encoded form (name) it will take at least 19 bits, e.g. 16 compartments and 7 clearance levels for each security class. In such a case parallel hardware could be used to check the permissibility of each attempted access without significant time overhead. However, if the files (repositories) had only say 36 bits of information this would cause an intolerable storage overhead. Because of fixed storage requirements for each security class value, storage overhead will be minimized by making the repository as large as possible.

On the other hand to obtain the total mediation required in a secure system there must be some mechanism which efficiently does the equivalent of checking current access privilege at every attempted access of a repository (file).

In a conventional system in which all accesses to the file system are under the control of a file system manager it would be reasonable to assign classification/compartment values to each file and to interpret the controller observe and modify type accesses of S_j to be reading and writing into the file system under the auspices of the file system manager. The security of information contained in the files could in theory be ensured by including security checking mechanisms in the file manager itself; e.g., a file could only be opened for reading by a process with clearance equal to or greater than the classification of the file. A major problem with this approach is that the security of information in files depends directly upon the continued integrity of the file system manager, the operation of which would have to be verified to be correct and to be guaranteed unmodifiable by malicious or errant programs. A second problem is the fact that after information was read into core the secure file system manager would have no control over sharing of information by processes in core.

The Multics approach appears to offer a feasible alternate solution in which mediation is rigorously imposed on all attempted accesses by the dynamic address computation mechanism built into the hardware. Under the Multics virtual memory all processor accesses to memory can be considered to be accesses to the file system and vice versa. There is no logical distinction between accessing a word in core, on

disk, or in the paging device. All accesses to storage in Multics are indirected through a segment descriptor word (SDW) in a segment descriptor table which is maintained for each process. Among other things the SDW contains the access privileges of the process for the segment to which it points. Correct operation depends upon maintaining the validity of the segment descriptor table for each process and upon the proper functioning of the supporting segmentation and paging control mechanisms. At this level we will concentrate on formalizing the behavior of the segment table, leaving the underlying mechanism as problems of implementation to be addressed in later more detailed specifications.

In the current Multics the segment descriptors are maintained on the basis of the access control lists (i.e. lists of permitted users) which are kept for each segment. We propose a similar procedure whereby decisions to update the segment descriptor table are based upon mandatory security restrictions. The present use of access control lists could be retained to provide discretionary "need-to-know" security.

In the following the term executor will be used to refer to the abstraction for process. Since we are considering only the problems of maintaining information security, the only aspects of the segment descriptors which we will discuss are the "read" and "write" bits which control the observe and modify type operations. These bits will be abstracted to two relations the view and alter relation which can be roughly interpreted to indicate whether or not a given file (segment) has been connected (i.e. a segment descriptor word has been

prepared for it and entered into the segment table of the appropriate executor). In order for information to move from a file to an executor the view relation must hold. This fact will be asserted as an axiom below. Upon the first attempt to view a given file a "missing segment-condition" will be raised which will cause explicit checking of mandatory security restrictions. If the attempted view is permissible then the view relation is set and thus any subsequent views are allowed without explicit checking. In this way total mediation can be effected with no additional overhead in time or storage except at initial attempted access. A similar argument applies to the "write" or alter type operation. While it is not clear how interprocess communication will be implemented, we have proposed similar constructs for accessing of mailboxes.

The dynamic or time-variant nature of an information processing system will be treated by considering the system to be a sequence of states. During each state the system will remain static as far as the security relevant conditions are concerned. Thus during a state there will be no new files or executors created, nor any connection etc. changed. Ordinary computation and movement of data will proceed as usual during the state. A transition to a next state will occur whenever some security relevant change in the system is required. By the nature of the transitions it cannot be expected that S_1 type axioms will hold from one state to the next. What we can hope to show is that if we start in a state which is S_1 -secure and if we restrict the transition to a subset of security preserving transitions then every state which can be reached through a sequence of secure transi-

tions will leave the system in a secure condition, that is, no unauthorized disclosures will ever be allowed.

Intuitively, the method of attack is to consider the several states with their component objects and relations to be combined with appropriate inter-state relationships into a single S_1 structure. Then it must be shown that with reasonable interpretation this new construct is secure and that the security thus displayed applies not only to data transfer paths within individual states but also to extended dynamic transfer paths between states.

4.2 A Formal Description of a Dynamic Secure System

In this section we shall propose an axiomatic system which allows us to describe the time-variant behavior of a secure information processing system. At any given time we assume the new system has a structure similar to S_1 in that it has sets of files, mailboxes and executors (agents) with corresponding relationships and functions, but with the additional flexibility that the said relations and functions may vary with time. We shall assume only that the set of security classes and its pre-ordering is constant. Thus we will allow additions or deletions to any of the sets of files, mailboxes and executors. Also the view, alter, block and wakeup relationships may be enlarged (i.e. new connections made) or diminished (disconnections). The classification and clearance functions will also be allowed to change.

In order to describe the current "state" of the system at any given time we shall compose all security relevant, time variant information about the system into a substructure called the state of the system.

For convenience we introduce the auxiliary set, states, and record the state information through use of additional relation and functions. Rather than to give an extensive list of axioms for S_2 we prefer to describe it as a nondeterministic automaton, that is, in terms of possible state values, and transitions from state to state. We will define a class of permissible transitions with the property that if the S_2 - system starts in an initial state which is "secure" and makes only permissible transitions then it will always stay in a "secure" state

and furthermore any "dynamic" information transfer paths will be secure in the sense of S_1 . This latter property will be established by showing that the structure formed by all states which are reachable by permissible transitions from some initial secure state is in fact an example of S_1 .

Finally, an example set of primitive permissible transitions (security events) will be given from which, we claim, any permissible transitions can be composed. It will be possible to derive formal specifications for the primitive transitions, and it is these specification along with a definition of an initial secure state which will be used in subsequent formalization.

At this level of abstraction we are not addressing the concomitant problems involved when the transitions are assumed to occur at the request of executors within the system. To the extent that state information is stored in classified files (or mailboxes) the state changes are modifications and must be governed by the usual restrictions on acquisition and dissemination of information. The solutions to those problems depend upon additional design decisions which are not necessary to answer the more general question raised in this chapter, namely, just what state conditions are necessary before a given transition is to be permitted.

The formal description of a dynamic secure system consists of the following structure:

$$S_2 = \langle E, F, M, C, S, \tau, \leq, \delta, \nu, \alpha, \beta, \omega, \text{clr}, \text{cls}, \overline{E}, \overline{F}, \overline{M}, \overline{\delta}, \overline{\nu}, \overline{\alpha}, \overline{\beta}, \overline{\omega}, \overline{\text{clr}}, \overline{\text{cls}} \rangle$$

E is a set of executors.

F is a set of files.

M is a set of mailboxes.

C is a set of security classes.

S is a set of states.

$\tau \subseteq S \times S$ the transition relation; thus, if $s \tau t$ we will say there is a transition from s to t where s and t are states.

$\leq \subseteq C \times C$ pre-ordering of C .

$\delta \subseteq F \times F$ the dominate relation on files.

$\nu \subseteq E \times F$ the view relation. ($e \nu f$ means executor e is connected to file f for viewing or reading information in the file.

We assume that under no other circumstances can an executor acquire information from a file.)

$\alpha \subseteq E \times F$ the alter relation. ($e \alpha f$ means executor e is connected to file f for writing, that is, the write bit is set.)

$\beta \subseteq E \times M$ the block relation. ($e \beta m$ means that executor e can interrogate mailbox m and if no message is available wait until one is received. In current implementation there is no counterpart

to the segment descriptor word for mailboxes (or event channels); it is considered sufficient to have the "name" of a mailbox in order to use it. For security it will be necessary to check classification/clearance restrictions before disclosing the "name" or else explicit checks will be necessitated at each requested access.)

$$\omega \subseteq E \times M$$

the wakeup relation. ($e \omega m$ means executor e can send a message to another executor which is blocked on mailbox m or will block on m at some later time.)

$$clr \subseteq E \times C$$

the clearance relation. (Since any executor may have different clearances in different states clearance is no longer a functional relation. For security it will be necessary to demonstrate that each executor has at any single time exactly one clearance.)

$$cls \subseteq F \cup M \times C$$

the classification relation. (Similar to clr above, it relates each file or mailbox to a set of security classes which it may have at different times.)

In addition to the foregoing sets and relations which are static throughout the life of the system we must also have relations which will indicate what the system looks like in each state. While this could be described by a single function from the set S into a set of state values each of which would be a fairly complex substructure we prefer to decompose the function in the following way:

$\bar{E}: S \rightarrow P(E)$ gives the subset of executors which exist in each state. (Thus, $\bar{E}(s)$ is the set of executors in the system during state s . We shall frequently use the notation E_s for the set $\bar{E}(s)$. Note $E = \bigcup_{s \in S} E_s$.)

$\bar{F}: S \rightarrow P(F)$ gives set of files current in each state. Again we shall use F_s to denote the set $\bar{F}(s)$.

$\bar{M}: S \rightarrow P(M)$ gives the set of mailboxes in each state.

$\bar{\delta}: S \rightarrow P(F \times F)$ gives the dominate relation in each state.

$\bar{v}: S \rightarrow P(E \times F)$ gives the view relation in each state.

$\bar{\alpha}: S \rightarrow P(E \times F)$ gives the alter relation in each state.

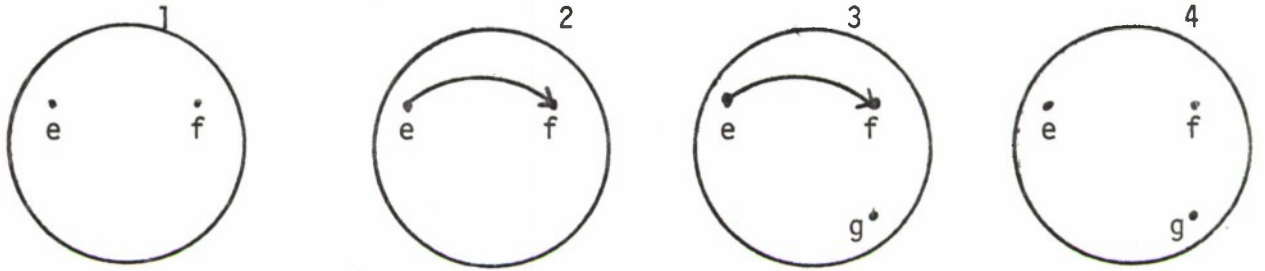
$\bar{\rho}: S \rightarrow P(E \times M)$ gives the block relation in each state.

$\bar{\omega}: S \rightarrow P(E \times M)$ gives the wakeup relation in each state.

$\bar{clr}: S \rightarrow P(E \times C)$ gives the clearance function in each state.

$\bar{cls}: S \rightarrow P(F \cup M \times C)$ gives the classification function in each state.

We now claim we have a framework in which state changes can be effeciently discussed. Let us illustrate this claim by a simple example. Suppose there is an S_2 -system which passes through the following states:



In "1" there is one executor e and one file f - no security classes in this example. In "2" e has been connected to f for viewing. Then in "3" a new file g has been added, and finally in "4" e has been disconnected from f . Let $S = \{S_1, S_2, S_3, S_4\}$ be a set of states for the system. Then $\bar{F}: S \rightarrow P(F)$ where $F = \{f, g\}$ is defined so that $\bar{F}(S_1) = \{f\}$, $\bar{F}(S_2) = \{f\}$, $\bar{F}(S_3) = \{f, g\}$ and $\bar{F}(S_4) = \{f, g\}$. Thus $\bar{F}$ gives all information about the set of files in each state. $\bar{v}: S \rightarrow P(\{e\} \times F)$ is defined by $\bar{v}(S_1) = \bar{v}(S_4) = \emptyset$ and $\bar{v}(S_2) = \bar{v}(S_3) = \{\langle e, f \rangle\}$. Now since all the sets, relations and functions are static, the only effect of a state change is that the "current" state changes. Hence, in discussing the dynamic behavior of the system we need only be concerned with the set S and the transition relation τ which indicates which state changes can occur.

For a given $s \in S$ we shall define the 10-tuple $\langle \bar{E}(s), \bar{F}(s), \bar{M}(s), \bar{\delta}(s), \bar{v}(s), \bar{\alpha}(s), \bar{\beta}(s), \bar{\omega}(s), \bar{clr}(s), \bar{cls}(s) \rangle$ to be an S_2 -state and shall usually write it as $\langle E_s, F_s, M_s, \delta_s, v_s, \alpha_s, \beta_s, \omega_s, cls_s, cls_s \rangle$ or even as $\langle E, F, M, \underline{\alpha}, v, \alpha, \beta, \delta, clr, cls \rangle_s$.

An S_2 -transition is an ordered pair of S_1 -states $\langle \langle E_s, F_s, M_s, \delta_s, v_s, \alpha_s, \beta_s, \omega_s, clr_s, cls_s \rangle, \langle E_t, F_t, M_t, \underline{\alpha}_t, v_t, \alpha_t, \beta_t, \omega_t, clr_t, cls_t \rangle \rangle$ such that

s t.

The S_2 -state gives a complete picture of the condition of all time-variant elements of the system at any time. We note that each S_2 -state has a structure similar to S_1 with the exception of the set of security classes and its accompanying preordering. We make use of the similarity to get the definition:

An S_2 -state is statically secure if and only if it satisfies the following axioms:

- A2.1 For all $e \in E_S$, $f \in F_S$, $e \nu_S f$ implies $cls_S(f) \leq_S clr_S(e)$.
- A2.2 For all $e \in E_S$, $m \in M_S$, $e \beta_S m$ implies $cls_S(m) = clr_S(e)$.
- A2.3 For all $e \in E_S$, $f \in F_S$, $e \alpha_S f$ implies $clr_S(e) \leq cls_S(f)$.
- A2.4 For all $e \in E_S$, $m \in M_S$, $e \omega_S m$ implies $clr_S(e) \leq cls_S(m)$.
- A2.5 For all $f \in F_S$, $f \delta_S f$.
- A2.6 For all $f, g \in F_S$, $f \delta_S g$ and $g \delta_S f$ implies $f = g$.
- A2.7 For all $f, g, h \in F_S$, $f \delta_S g$ and $g \delta_S h$ implies $f \delta_S h$.
- A2.8 For all $f, g, h \in F_S$, $f \delta_S g$ and $h \delta_S g$ implies $f \delta_S h$ or $h \delta_S f$.
- A2.9 For all $e \in E_S$, $f, g \in F_S$, $e \nu_S g$ and $f \delta_S g$ implies $e \nu_S f$. (That is, in order to be connected for viewing a file, an executor must be connected to view all files which dominate it.)
- A2.10 For all $e \in E_S$, $f, g \in F_S$, $e \alpha_S g$ and $f \delta_S g$ and $f \neq g$ implies $e \nu_S f$. (To alter a file, an executor must be connected to view all files which strictly dominate it.)
- A2.11 For all $f, g \in F_S$, $f \delta g$ implies $cls_S(f) \leq cls_S(g)$.

Definition An S_2 -transition $\langle s, t \rangle$ is called preserving if and only if s is statically secure implies t is statically secure. We further define the relation τ to be preserving if and only if for all $s, t \in S$, $s \tau t$ implies $\langle s, t \rangle$ is preserving. Hence, if τ is preserving, s is statically secure and $s \tau t$, then t is statically secure.

In the following we shall be concerned with the security preserving properties of sequences of transitions. In general the relation τ is not transitive, in particular, for the set of primitive transitions to be introduced later it is not true that the composition of two primitive transitions is a primitive transition. The next lemma will show that compositions of preserving transitions are preserving.

Lemma 4.2.1: If τ is security preserving, s is a statically secure state, and $s \tau^* t$, then t is a statically secure state.

($\tau^* = \bigcup_{n=0}^{\infty} \tau^n$ is the transitive, reflexive closure of τ .)

Proof: (Using induction) Let N be the set of natural numbers, and let $P(n)$ be the property " τ^n is preserving". Then we let $M = \{n \in N \mid P(n) \text{ is true}\}$, and use induction to prove $M = N$.

First: $0 \in M$ since $P(0)$ holds i.e. τ^0 is preserving because $s \tau^0 t$ implies $s = t$.

Second: suppose $n \in M$ then τ^n is preserving, but $\tau^{n+1} = \tau^n \circ \tau$. Then if $s \tau^{n+1} t$ and s is statically secure there exists a $u \in S$ such that $s \tau^n u$ and $u \tau t$. By inductive hypothesis τ^n is preserving; thus, u is statically secure and since τ itself is preserving t is also

statically secure. Hence τ^{n+1} is preserving, $P(n+1)$ holds, and $n+1 \in M$. By induction $N=M$ and $P(n)$ holds for all n , or τ^n is preserving for all n .

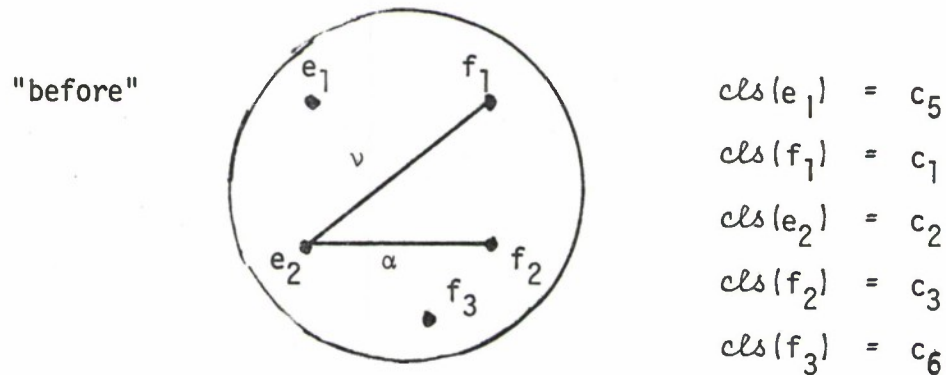
Last: Suppose s is statically secure and $s \tau^* t$; then, $s \tau^n t$ for some n so that t is statically secure. 

We shall say that a state t is reachable from a state s if and only if $s \tau^* t$. Immediately we have:

Corollary 4.2.1: If τ is preserving, and state s is statically secure, then every state reachable from s is statically secure. 

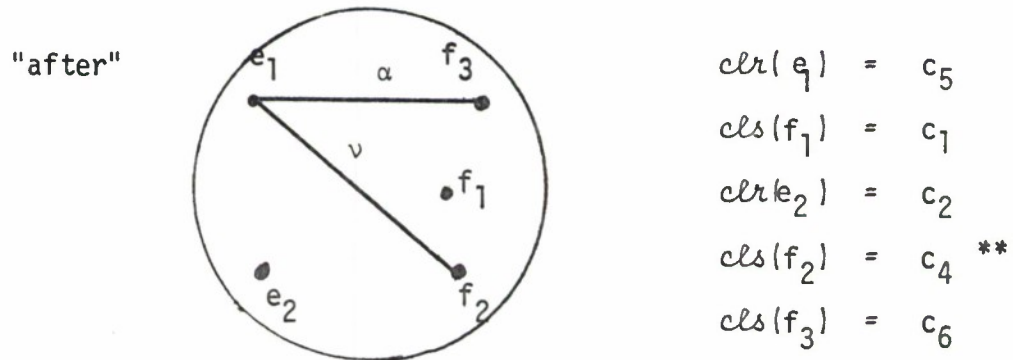
4.3 An Informal Discussion of Some Dynamic Problems

It might appear that this last result gives a sufficient restriction on the transitions to ensure a "secure" S_2 ; however, this is not the case as is shown in the following example. Consider the system with executors $\{e_1, e_2\}$ and files f_1, f_2, f_3 with security classes $\{c_1, c_2, c_3, c_4, c_5, c_6\}$ such that $c_1 \trianglelefteq c_2 \trianglelefteq c_3$, $c_4 \trianglelefteq c_5 \trianglelefteq c_6$; c_3 and c_4 are not comparable. Now suppose that the following state occurs:



We claim this "before" state is statically secure because $e_2 \vee f_1$ implies $cls(f_1) \trianglelefteq cls(e_2)$, $c_1 \trianglelefteq c_2$ holds; and $e_2 \alpha f_2$ implies $cls(e_2) \trianglelefteq cls(f_2)$, $c_2 \trianglelefteq c_3$ holds. The only information transfer path is from f_1 to f_2 , and this does not allow unauthorized disclosure since $cls(f_1) \trianglelefteq cls(f_2)$.

Then suppose there is a transition to the state:



We claim this "after" state is also statically secure because $e_1 \alpha f_3$ implies $clr(e_1) \sqsubseteq cls(f_3)$, $c_5 \sqsubseteq c_6$ holds; and $e_1 \gamma f_2$ implies $cls(f_2) \sqsubseteq clr(e_1)$, $c_4 \sqsubseteq c_5$ holds. There is a transfer path from f_2 to f_3 but this is allowed since $cls(f_2) \sqsubseteq cls(f_3)$.

Both the "before" and "after" states are statically secure, the transition is preserving, and everything seems to be in good order. But if we look more carefully we see that information which may have been transferred from f_1 to f_2 in the "before" state may now be transferred from f_2 to f_3 while $cls(f_1)$ is not comparable with $cls(f_3)$. Although both states are statically secure, the temporal composition contains an information transfer path from f_1 to f_3 while it is not true that $cls(f_1) \sqsubseteq cls(f_3)$.

What we have just seen is an example of a preserving transition which leads to a possible non-secure condition; hence, the restriction to preserving transitions is not sufficient for a system which is "dynamically secure". Before we attempt to further qualify the transitions, we need a formal definition of what it means to be "secure". S_1 -security is precisely a formalization of security which prohibits unauthorized disclosures by controlling implicit data transfer paths. We will be satisfied that S_2 is secure if it can be shown to be an example (or valid interpretation) of S_1 , and this is the next step in our development, that is, to show that with some additional restrictions on S_2 -transitions we get a structure which is S_1 -secure.

We need a treatment in which the "dynamic" data transfer path in our preceding example is detected. A fundamental problem is that file f_2 in the example forms a link in two different information transfer

paths, the existence of which depend upon two different states. A possible solution is to flag the elements in the two states so that f_1^b in the "before" state is logically distinct from f_1^a in the "after" state, so also for the other elements. With this distinction it is possible to picture the two states together as in the following figure.

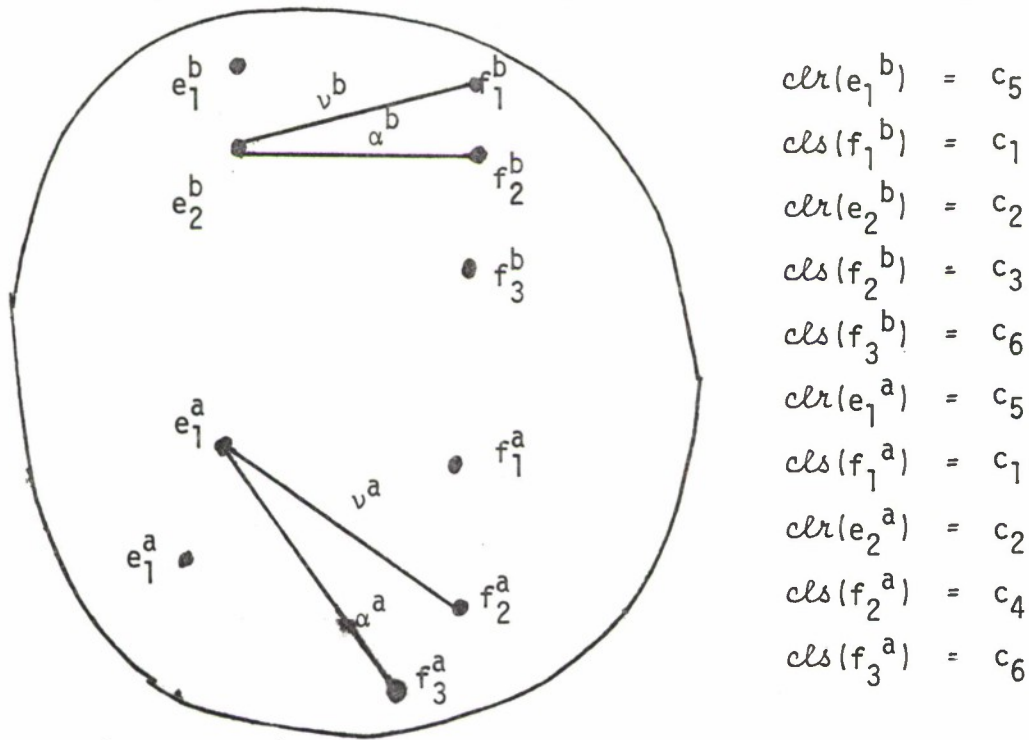


Figure 4.3.1

While in the earlier presentation $cls(f_2) = c_3$ or c_4 depending on state, classification is now a function as is required in S_1 . Files f_1^b and f_1^a are incarnations of the same "real" file. From what we know about real files we believe that any information stored in f_1^b will remain in it through the transition and can be retrieved from f_1^a . But since there is no formal connection between f_1^b and f_1^a , additional structure is necessary. A file to file transfer is needed, but this is not provided for in S_1 , and clearly there is no actual transfer of data since f_1^b and f_1^a are merely two names for the same file.

Before proposing our solution let us digress slightly to present a possible alternative. It is the function of executors to transfer information; thus, one way to maintain the information content across transitions would be to hypothesize a new class of agents, perhaps called quasi-executors, whose function is to retrieve information from files in the "before" state and store into the corresponding file in the "after" state. The file-to-file transfer logically becomes an information transfer path which can be handled in S_1 theory. An interesting result is that if we require quasi-executors to conform to the normal classification/clearance restrictions, then the classification of a file in the "after" state must be greater than or equal to the classification of its counterpart in the "before" state. Incidentally, the failure to satisfy this requirement is the source of the insecurity of the previous example. The quasi-agent treatment was rejected because it also leads to quasi-files and quasi-mailboxes and it appears to be rather unnatural or nonintuitive. We now return to the mainline of development by introducing a device which is both easier to formalize and we hope more natural.

Referring to Fig.4.3.1 we note that if executor e_2^b alters file f_2^b , the effect of the alteration should persist in file f_2^a so that any executor which can view f_2^a should thereby be able to "view" the effects of the alteration of e_2^b . The resulting principle of "persistence of information" will be generalized to imply that if an executor can alter a file, then since the alteration persists potentially as long as the file exists it can effectively alter every subsequent incarnation of that file. This principle implies that classifications of files

must increase or remain constant since alteration must always act upon a file of equal or higher security class than that of the acting executor.

On the other hand, e_1^a 's being able to alter f_3^a does not imply that it can also alter f_3^b for this would mean that an executor could effect a change in a file in a state which no longer existed. We shall, however, permit executors to carry information through transitions. Hence, in the example, e_1^a can alter f_3^a in ways which reflect information acquired in state "before" by e_1^b . In this manner e_1^b can alter f_3^a . This may sound counterintuitive, but all we are saying is that there exists the possibility of transfer of information from e_1^b to f_3^a without an intervening file or executor.

Similar arguments show that it is reasonable to assume that in the case of the view relation an executor cannot view a file in a future state-principle of "non prescience of executors" - but can in effect view files in past states. By persistence of information, once an executor has "viewed" a file the action of any successor of that executor may reflect the viewed information.

Another principle is that if an executor and a file do not both exist in at least one common state there can be no direct transfer of information (neither view nor alter).

In Fig. 4.3.2 we have a representation of an executor e and two files f and g which exist through several states. The horizontal transition lines can be thought of as cutting the executor into a sequence of entities each of which carries the name of the executor it represents and the name of the state in which it occurs. The files

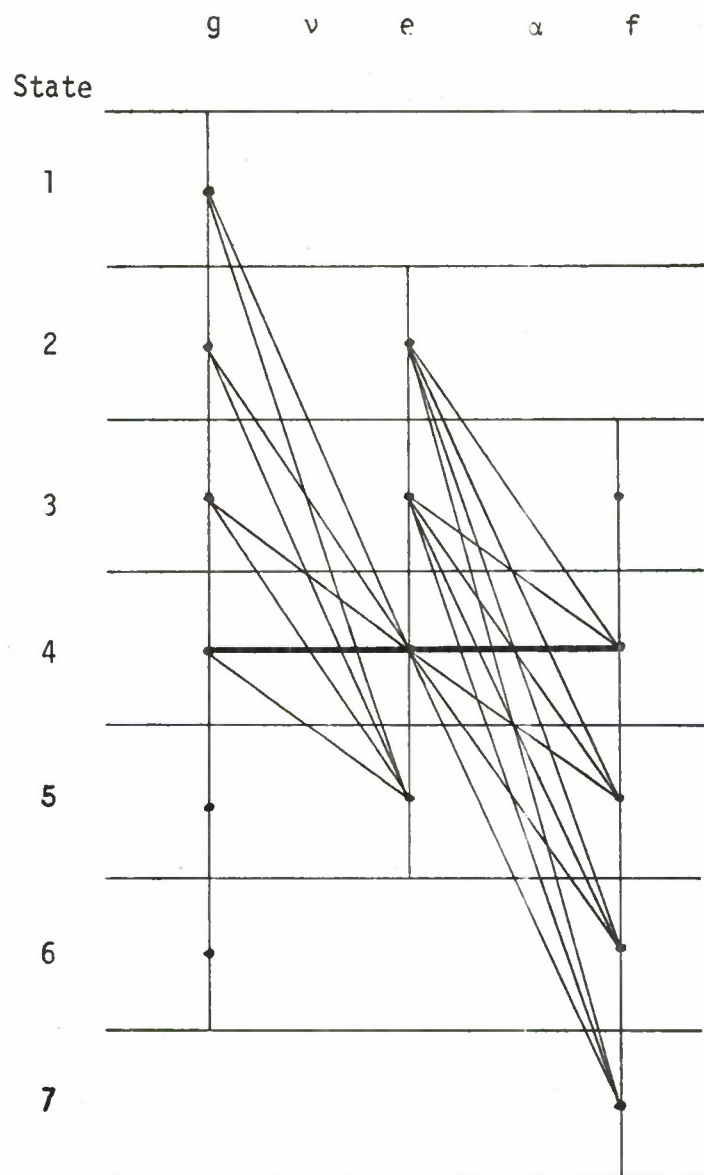


Figure 4.3.2

are similarly decomposed. The names transient-file and transient-executor have been suggested for these new constructs. If we suppose that e is connected to alter f in state "4" then the effect of possible alteration of f will persist until f is deleted after state "7". Also, information available to e in states "2" and "3" may be transferred into f . Furthermore, if e is also connected to view g in state "4", we get the additional direct transfer path so illustrated. We claim that we thus include all possible such paths. That is to say that if information moves at all it must move along the lines shown. This procedure will be formalized later after some additional notation has been introduced. For now let us review the example of Fig. 4.3.1.

In Fig. 4.3.3 the "before-after" picture is repeated with implied inter-state relations added. We now see that there is an information transfer path from f_1^b to f_3^a namely: $(f_1^b, e_2^b, f_2^b, e_1^a, f_3^a)$ (also $\langle f_1^b, e_2^b, f_2^a, e_1^a, f_3^a \rangle$). And since it is not true that $cls(f_1^b) \leq cls(f_3^a)$ we better not be able to show that such a structure is an example of S_1 . Clearly, additional restrictions on the transitions are necessary. Although the explicit view relation between e_1^a and f_2^a is allowable, the implied view between e_1^a and f_2^b should be prohibited since it is not true that $cls(f_2^b) \leq cls(e_1^a)$. After all this, it turns out that it is sufficient to require that if classifications (and clearances) change they must increase. Thus in the example we should require that $cls(f_2^b) \leq cls(f_2^a)$ or else not permit the transition. Of course under such requirements we have $cls(f_1^b) \leq cls(f_2^b) \leq cls(f_2^a) \leq cls(f_3^a)$ and we are in good shape. We will next show that by generalizing the preceding arguments we can show that all possible dynamic information

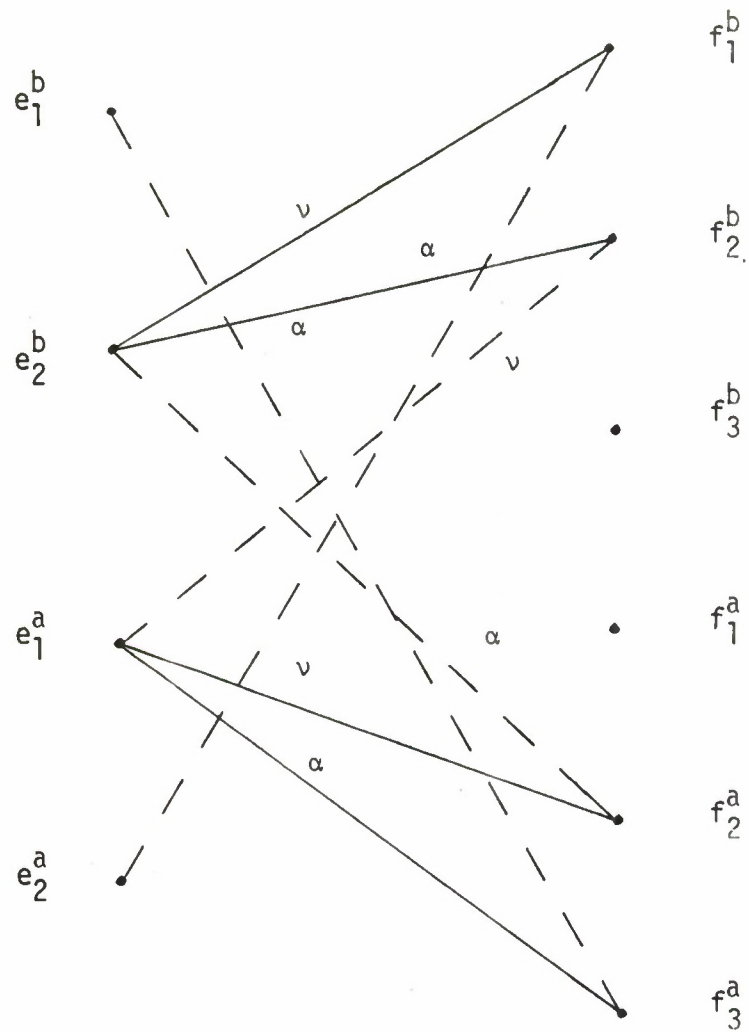


Figure 4.3.3

transfer paths behave in proper S_1 fashion.

4.4 The Dynamic Security of S_2

Given an S_2 -structure we shall by the following construction form an associated structure of transient-executors, transient-files etc. in which each S_2 object is fragmented into logically distinct components in each state. The term transient is used to indicate the transitory existence of the new object. The temporal decomposition is accomplished by forming for each object in S_2 the cartesian product of itself and the set of states in which it exists. While some of the relations between the transient objects are natural and straight-forward, we shall have to elaborate upon the information moving relations to be sure that all possible S_2 dynamic information transfer paths are included.

$$(4.4.1) \quad E_S = \{ \langle e, s \rangle \mid s \in S \text{ and } e \in E_S \}.$$

We begin with the set of transient-executors; thus, $\langle e, s \rangle \in E_S$ means that e is an executor in state s . Should $\langle e, t \rangle$ also belong to E_S , then e is also an executor in state t .

$$(4.4.2) \quad F_S = \{ \langle f, s \rangle \mid s \in S \text{ and } f \in F_S \}$$

The set of transient files.

$$(4.4.3) \quad M_S = \{ \langle m, s \rangle \mid s \in S \text{ and } m \in M_S \}$$

Transient mailboxes.

$$(4.4.4) \quad \delta_S \subseteq F_S \times F_S$$

is the dominates relation on transient-files. In terms of the S_2 dominates relation it is defined by $\langle f, s \rangle \delta_S \langle g, t \rangle$ if and only if $s=t$ and $f \delta_S t$. We do not extend the dominates

relation across transitions. Thus we shall not allow one transient file to dominate another in a different state.

$$(4.4.5) \quad \text{clr}_S: E_S \rightarrow C$$

is the clearance function defined by for all $\langle e, s \rangle \in E_S$, $\text{clr}_S(\langle e, s \rangle) = \text{clr}_S(e)$.

To have S_1 -security we must demonstrate that it is indeed a function. But this follows from the definition and from the fact that for each $s \in S$, $\text{clr}_s: E \rightarrow C$ is a function--part of the requirement for a statically secure state.

$$(4.4.6) \quad \text{cls}_S: F_S \cup M_S \rightarrow C$$

is the classification function defined by for all $\langle x, s \rangle \in F_S \cup M_S$, $\text{cls}_S(\langle x, s \rangle) = \text{cls}_S(x)$,

It will be convenient to introduce some additional notation in the form of a relation defined on the transient objects as follows.

For all $\langle x, s \rangle, \langle y, t \rangle \in E_S \cup F_S \cup M_S$ we shall say $\langle x, s \rangle \pi \langle y, t \rangle$ (i.e. $\langle x, s \rangle$ is perpetuated by $\langle y, t \rangle$) if and only if $x=y$ and $s \tau t$.

Thus a transient-file $\langle f, s \rangle$ $\langle f, t \rangle$ is perpetuated by a transient-file if there is a transition from s to t , and f exists both in states s and t . This provides a means for discussing the passing along of information from transient-file to transient file and also for transient-executors and transient-mailboxes.

The relations view, alter, block and wakeup must be constructed on the transient-objects so that all possible information transfer

paths are included. A simple example was given in sec. 4.3. For example, the view relation must encompass all direct transfers from transient-files to transient-executors. By "direct" we mean the information does not pass through any intervening file or executor; thus, we see that there can be no direct transfer unless the file and executor coexist in some common state. Furthermore, information can be transferred only if in that common state the executor is view connected to the file.

We formally state this as the

Principle of Concurrency: If $\langle e, s \rangle v_S \langle f, t \rangle$, then there must be a state $u \in S$ such that $e \in E_u$, $f \in F_u$, and $e v_u f$. Note the notation $v_S \subseteq E_S \times F_S$ for the transient view relation.

We also have the

Principle of persistence of information: we assume that information in a file may persist as long as the file exists; thus, if $\langle e, s \rangle v_S \langle f, t \rangle$ and $\langle f, u \rangle \pi \langle f, t \rangle$, then must have $\langle e, s \rangle v_S \langle f, u \rangle$. Furthermore, the repeated application of this principle leads to a similar statement in terms of $\pi^* = \bigcup_{n=0}^{\infty} \pi^n$ the transitive, reflexive closure of π . Finally we state the

Principle of nonprescience: an executor cannot perceive (and thus pass on or make decisions on the basis of) information in a file before the transition at which the view connection is made. We are saying that views also

move information forward in time. Similar arguments apply to alter, block and wakeup.

On the basis of the foregoing we make the following definition.

$$(4.4.7) \quad v_S \subseteq E_S \times F_S \quad \text{the transient view relation defined by}$$

$\langle e, s \rangle v_S \langle f, t \rangle$ if and only if there exists $u \in S$ such that $e \in E_u$, $f \in F_u$, $e v_u f$, and $\langle f, t \rangle \pi^* \langle f, u \rangle$ and $\langle e, u \rangle \pi^* \langle e, s \rangle$. Thus for information to be transferred from f in state t to e in state s it is necessary that e and f exist in state u , that $e v_u f$ and that state u be reachable from state t and state s reachable from state u .

Similarly for other relations we have:

$$(4.4.8) \quad \alpha_S \subseteq E_S \times F_S \quad \text{the transient alter relation defined by}$$

$\langle e, s \rangle \alpha_S \langle f, t \rangle$ if and only if there exists $u \in S$ such that $e \in E_u$, $f \in F_u$, $e \alpha_u f$, and $\langle f, u \rangle \pi^* \langle f, t \rangle$ and $\langle e, s \rangle \pi^* \langle e, u \rangle$.

$$(4.4.9) \quad \beta_S \subseteq E_S \times M_S \quad \text{the transient block relation defined by}$$

$\langle e, s \rangle \beta_S \langle m, t \rangle$ if and only if there exists $u \in S$ such that $e \in E_u$, $m \in M_u$, $e \beta_u m$, and $\langle m, t \rangle \pi^* \langle m, u \rangle$ and $\langle e, u \rangle \pi^* \langle e, s \rangle$.

$$(4.4.10) \quad \omega_S \subseteq E_S \times M_S \quad \text{the transient wakeup relationship defined by}$$

$\langle e, s \rangle \omega_S \langle m, t \rangle$ if and only if there exists $u \in S$ such that $e \in E_u$, $m \in M_u$, $e \omega_u m$, and $\langle m, u \rangle \pi^* \langle m, t \rangle$ and $\langle e, s \rangle \pi^* \langle e, u \rangle$.

To show that the system of transient-objects is S_1 -secure we will need to show among other things that:

for all $\langle e, s \rangle \in E_S$, and $\langle f, t \rangle \in F_S$, $\langle e, s \rangle \nu_S \langle f, t \rangle$ implies that $cls_S(\langle f, t \rangle) \leq clr_S(\langle e, s \rangle)$.

The proof breaks into three parts. By (4.4.7) we have an intermediate state u in which $e \nu_u f$. Now if we restrict the set of states to those which are statically secure, u being statically secure implies $cls_u(f) \leq clr_u(e)$ which under definitions (4.4.5) and (4.4.6) yields $cls_S(\langle f, u \rangle) \leq clr_S(\langle e, u \rangle)$. Secondly if we can be sure that $\langle f, t \rangle \pi^* \langle f, u \rangle$ implies $cls_S(\langle f, t \rangle) \leq cls_S(\langle f, u \rangle)$ and that $\langle e, u \rangle \pi^* \langle e, t \rangle$ implies $clr_S(\langle e, u \rangle) \leq clr_S(\langle e, t \rangle)$ and thirdly, if we know $\leq$ is transitive, then we can conclude that $cls_S(\langle f, t \rangle) \leq clr_S(\langle e, s \rangle)$. With this motivation we make the following definitions.

An S_2 transition $\langle s, t \rangle$ is permissible if and only if it is preserving and satisfies the following axioms:

A2.12 For all $e \in E$, $e \in E_S \cap E_t$ implies $clr_S(e) \leq clr_t(e)$.
(That is if e exists in states s and t and $s \tau t$, then the clearance e can only increase if it changes.)

A2.13 For all $f \in F$, $f \in F_S \cap F_t$ implies $cls_S(f) \leq cls_t(f)$.

A2.14 For all $m \in M$, $m \in M_S \cap M_t$ implies $cls_S(m) \leq cls_t(m)$.

(These three axioms were shown to be necessary in the examples).

τ is permissible if and only if for all $s, t \in S$, $s \tau t$ implies $\langle s, t \rangle$ is permissible.

An S_2 -structure (see p.48) will be called S_2 -secure or dynamically secure if and only if it has the following properties:

1. τ is permissible.
2. There is an initial state s_0 in S which is statically secure.
3. $s \in S$ implies s is reachable from s_0 .
- (A2.15) 4. For all $c \in C$, $c \preceq c$. ($\preceq$ is reflexive.)
- (A2.16) 5. For all $c, d, e \in C$, $c \preceq d$ and $d \preceq e$ implies $c \preceq e$.
($\preceq$ is transitive).
- (A2.17) 6. For all $f, g \in F$, $s, t \in S$, $f \delta_s g$ and $g \in F_t$ implies $f \in F_t$ and $f \delta_t g$, which states that while a file g exists all of the files which dominate g must continue to exist and continue to dominate g .

The major result of the chapter is showing that the system of transient objects constructed in an S_2 -secure structure is an example of S_1 . The resultant properties of the transients allow us to make assertions about information transfer paths in S_2 itself. Since the transient-structure lies between S_1 and S_2 it might well be called $S_{1.5}$.

$S_{1.5} = \langle E_S, F_S, M_S, C, \preceq_S, \nu_S, \alpha_S, \beta_S, \omega_S, clr_S, cls_S, \pi \rangle$ where the components are as defined above.

Proposition 4.4.1: The $S_{1.5}$ structure associated with an S_2 -secure structure is an S_1 -secure structure.

Proof: The details of the proof are given in Appendix C.

An interpretation of the basic theorem of S_0 is provable in $S_{1.5}$.

Thus we have:

Theorem 4.4.1: In an $S_{1.5}$ -structure associated with an S_2 secure structure if there is an information transfer path

from transient-file (or mailbox) $\langle f, s \rangle$ to transient-file or (mailbox) $\langle g, t \rangle$, then $cls_S(\langle f, s \rangle) \leq cls_S(\langle g, t \rangle)$.

In the language of S_2 this can be stated as

Corollary 4.4.1: In an S_2 -secure structure if there is a dynamic information transfer path from file f in state s to a file g in state t , then the classification of f (in state s) is less than or equal to the classification of g (in state t).

In conclusion, if we have an S_2 -structure which has an initial statically secure state, and if we allow only transitions which preserve static security, maintain classification from state to state, then there will be no implicit information transfers which could lead to unauthorized disclosure.

5. THE SECURITY EVENTS IN S_3

5.1 Introduction

In the foregoing chapter, we evolved a set of axioms which are sufficient to ensure a dynamically secure system. The S_2 -structure is described as a system of S_1 -like states with an accompanying set of permissible transitions which indicate what sequences of states may be allowed in a secure system. The representation is highly abstract in that files, mailboxes, and executors are considered to be points without internal structure. However, if we look at our goal, a formal specification of a Multics-like system, we see that each of these elements is by itself quite complex. To move towards this more complex representation, we will now present a state space which more accurately depicts the real target system. The behavior of the new system will be formally described by a set of security events or elementary operations. After the events have been presented it will be shown in the next section that the events constitute a set of permissible transitions because all S_2 -axioms are satisfied.

As in the more abstract description the state of the system includes the sets of files, mailboxes, and executors which are current, their classification (or clearance) and the view, alter, block, and wake up relations in effect at the time. Now, the classification and status (present or not) of a file are only two of many attributes. A file also has length, location, authorship, and so forth. To the extent that these secondary attributes can be changed by executors and those changes perceived by other executors, they must be considered possible information channels. In general the attributes of a file will be kept

in another file -- usually the dominating file in the file hierarchy -- so that the information will be naturally protected by the security mechanism.

Since we cannot hope to anticipate all conceivable file attributes, we introduce a new set.

V_F : the set of all possible file attributes. (These attributes will include such things as length, contents, classification, etc.)

To indicate which value pertains at any given time we introduce a time-variant function.

$Fav: F \rightarrow V_F$ is the file attribute function.

A change in any file attribute is then reflected by a change in Fav . Present value of a particular attribute, such as classification, will be singled out by a function defined on V_F , such as $cls: V_F \rightarrow C$ for file classification. Thus $cls(Fav(f))$ would give the value of the classification attribute of file f , or in notation of Chapter 4 $cls_s(f) = cls(Fav(f))$ where s is the current state. In this chapter where we need be concerned only with the current state and the next state (or old and new states) we shall drop state subscripts and adopt the convention that primes indicate next state. For example $Fav(f) = Fav'(f)$ means there was a change to a new state, but the file attributes of f remained the same.

Similar sets and functions are introduced for mailboxes and executors.

V_M : the set of possible values of all mailbox attributes. (classification, contents, etc.)

Pr : the set of all possible values which all the properties of an executor might have. (For example, clearance, priority, etc.)

$Mav: M \rightarrow V_M$ is the mailbox attribute value function.

$E_p: E \rightarrow Pr$ is the executors property function.

There are also several functions which return specific properties (or attributes as they will be called in the next chapter).

$st: V_F \rightarrow \{USED, UNUSED\}$ is the "status" function. It will often be used in conjunction with Fav and hence we define the following shorthand:

$ST: F \rightarrow \{USED, UNUSED\}$ and is equivalent to $Fav \circ st$, a composite function. In a similar vein, the "classification" function:

$CLS: F \rightarrow C$ is equivalent to $Fav \circ cls$.

$CLR: E_p \rightarrow C$ the "clearance" function is equivalent to $Fav \circ clr$.

$Dat: Pr \rightarrow Info$ the "data" function returns information which has been obtained by the executor.

$Con: V_M \rightarrow Info$ the "contents" function returns the information held by the specified mailbox.

In the following section we shall catalogue the security events in the system.

5.2 Event Descriptions

E1 Executor e becomes view-connected to file f.

This event establishes permission for executor e to view file f. However, before this operation is allowed to have any effect, certain conditions must hold.

C1.1 $ST(f) = USED$.

That is, file f must be in use since it doesn't make sense to connect to a nonexistent file.

C1.2 $CLS(f) \leq CLR(e)$

The clearance of the executor must be greater than or equal to the classification of f.

C1.3 $e \vee d$

File f's parent must be view connected since the directory hierarchy must be searched in order to find f. This condition is sufficient to guarantee that all dominating files can be viewed.

Now we must assert exactly what the operation does.

P1.1 $ST'(f) = ST(f)$

The file's status must remain USED.

P1.2 For all f_1 in F , $CLS'(f_1) = CLS(f_1)$

That is, the classifications of all files are unchanged.

P1.3 For all f_1 in F , $Fav'(f_1) = Fav(f_1)$

All other file attributes remain unchanged.

P1.4 For all e_1 in E , f_1 in F , $e_1 \delta' f_1$ if and only if $e_1 \delta f_1$.

In other words, the "dominates" relation remains constant. As will be seen, only the "create" and "delete" events have an effect upon δ .

P1.5 For all e_1 in E , f_1 in F , $e_1 \alpha f_1$ if and only if $e_1 \alpha f_1$.

The "can-alter" relation is unchanged, i.e. any executor can alter exactly those files which it could alter before the event.

P1.6 For all e_1 in E , f_1 in F , $e_1 \vee' f_1$ if and only if $e_1 \vee f_1$

OR $[e_1 = \underline{e} \wedge f_1 = f \wedge C1.1 \wedge C1.2 \wedge C1.3]$

That is, the requested addition is made to the "can-view" relation if the conditions are met. Otherwise, the "can-view" relation is unchanged.

P1.7 For all m_1 in M , $M\text{-CLS}'(m_1) = M\text{-CLS}(m_1)$

All mailboxes' classifications remain the same.

P1.8 For all m_1 in M , $Mav'(m_1) = Mav(m_1)$

The other mailbox attributes are also unchanged.

P1.9 For all e_1 in E , m_1 in M , $e_1 \beta' m_1$ if and only if $e_1 \beta m_1$.

The "can-block" relation is unchanged.

P1.10 For all e_1 in E , m_1 in M , $e_1 \omega' m_1$ if and only if $e_1 \omega m_1$.

The "can-wake" relation is unchanged.

P1.11 For all e_1 in E , $CLR'(e_1) = CLR(e_1)$.

The executors' clearances do not change.

Note: it will be seen that only one event - "Raise Clearance" can change this property.

P1.12 For all e_1 in E , $Ep'(e_1) = Ep(e_1)$.

All other executor properties are also unchanged.

As can readily be seen, most properties in this event were unchanged. For the remainder of this chapter, only those properties which are changed by the event in question will be listed. However, the properties will always be presented in the same order, and numbers

will be skipped for those properties not discussed. For example, the "can-wake" relation will always be presented as property Px.10, where "x" corresponds to the event number.

E2 Executor e becomes alter-connected to file f.

The second event is similar to the first one except that it is the "can-alter" relation which is changed.

C2.1 $ST(f) = USED$

The file must exist.

C2.2 $CLR(e) \triangleq CLS(f)$

In order for executor e to be able to alter file f, the information-dissemination axiom must be true; that is the file's classification must be greater than or equal to the executor's clearance.

C2.3 $e \vee d$

That is, e must be able to view file d - (f's parent directory).

Note: this condition guarantees that the directory hierarchy can be searched.

P2.5 For all e_1 in E , f_1 in F , $e_1 \alpha' f_1$ if and only if $e_1 \alpha f_1$
 OR $[e_1 = \underline{e} \wedge f_1 = \underline{f} \wedge C2.1 \wedge C2.2 \wedge C2.3]$

If the conditions are all satisfied, the requested addition is made to the alter relation, otherwise there is no change.

All other properties remain as they were before the occurrence of E2.

Note that a file can become "view and alter connected" if both events E1 and E2 occur successfully. As we have shown, any serial combination of events can occur consecutively without endangering security.

E3 Executor $\underline{e}$ becomes disconnected from file $\underline{f}$.

This event is used to reverse the effects of the two previous events. It would generally occur when a user logs out although it might also be invoked in the middle of a run to protect a data file from an undebugged program.

C3.1 $ST(f) = \text{USED}$

P3.5 For all e_1 in E , f_1 in F , $e_1 \alpha' f_1$ if and only if $e_1 \alpha f_1$
 $\wedge \neg ((e_1 = \underline{e}) \wedge (f \delta f_1) \wedge C3.1)$

In other words, all files in the subtree dominated by $\underline{f}$ will be "alter" disconnected from executor $\underline{e}$.

P3.6 For all e_1 in E , f_1 in F , $e_1 \nu' f_1$ if and only if $e_1 \nu f_1$
 $\wedge \neg ((e_1 = \underline{e}) \wedge (f \delta f_1) \wedge C3.1)$.

Similarly, the view relation is changed to disallow viewing of file $\underline{f}$ or its subtree by executor $\underline{e}$.

All other properties are unchanged.

E4 Executor e creates file f in directory d with attribute values V and classification c.

C4.1 $ST(d) = USED$

File f's parent directory is currently in use. This is necessary to guarantee that we have a tree structure.

C4.2 $CLR(e) \triangleq CLS(d)$

To avoid an illegal flow of information, the executor must be at a clearance which is less than or equal to the classification of the directory since creating a file includes making an entry in the directory.

C4.3 $d \delta f$ and for all f_1 in $F \rightarrow (d \delta' f_1 \wedge f_1 \delta' f \wedge f_1 = d)$

File f is in directory d.

C4.4 $ST(f) = UNUSED.$

File f is not currently in use.

C4.5 $CLS(d) \triangleq C$

The classification specified for the new file must be greater than or equal to d's classification. (This is a consequence of the tree-structure axioms of S_1).

C4.6 $\delta t(V) = USED$

The requested value for status must be "USED".

P4.1 For all f_1 in F , $C4.1 \wedge C4.2 \wedge C4.3 \wedge C4.4 \wedge C4.5 \wedge C4.6 \wedge f_1 = f$ implies $ST'(f) = \delta t(V)$ while $\neg(C4.1 \wedge C4.2 \wedge \dots \wedge C4.6 \wedge f_1 = \underline{f})$ implies $ST'(f_1) = ST(f_1)$.

If the conditions are satisfied, the status of f becomes "USED".

P4.2 For all f_1 in F , $CLS'(f_1) = \begin{cases} CLS(f_1) & \text{if } \neg(f = f_1 \wedge C4.1 \\ & \wedge C4.2 \wedge \dots \wedge C4.6) \\ \delta t(V) & \text{if } f = f_1 \wedge C4.1 \wedge \dots \wedge C4.6 \end{cases}$

The specified classification is assigned to file f if the conditions are met. Note that $CLS(f)$ (that is f 's classification before the event) is undefined.

P4.3 For all f_1 in F ,

$$Fav'(f_1) = \begin{cases} Fav(f_1) & \text{if } \neg(f_1 = f \wedge C4.1 \wedge \dots \wedge C4.6) \\ V & \text{if } f_1 = f \wedge C4.1 \wedge \dots \wedge C4.6 \end{cases}$$

Provided that the conditions are satisfied the other attributes are assigned as designated.

P4.4 a) For all f_1, f_2 in F , $f_1 \delta' f_2$ if and only if $f_1 \delta f_2$ OR

$$(f_1 \delta d \wedge (f_2 = f) \wedge C4.1 \wedge \dots \wedge C4.6) \text{ OR}$$

$$((f_1 = f_2 = f) \wedge C4.1 \wedge \dots \wedge C4.6)$$

b) For all f_1 in F , $f \delta f_1 \iff f = f_1$

The tree structure is changed to include $\underline{f}$. Also, $\underline{f}$ dominates only itself.

E5 Executor $\underline{e}$ destroys file $\underline{f}$ in directory $\underline{d}$

This event allows the deletion of unwanted segments and directories. Note that we do not want dangling subtrees due to A2.17. Therefore, if we wish to delete a directory, we insist that any file it dominates is also deleted.

C5.1 $ST(d) = \text{USED}$. The parent directory must be in use.

C5.2 $CLR(e) \leq CLS(d)$

We must be able to alter $\underline{d}$, since the directory hierarchy will no longer have a pointer to $\underline{f}$.

P5.1 For all f_1 in F ,

$$ST'(f_1) = \begin{cases} \text{UNUSED} & \text{if } f \delta f_1 \wedge C5.1 \wedge C5.2 \\ ST(f_1) & \text{if } \neg(f \delta f_1 \wedge C5.1 \wedge C5.2) \end{cases}$$

Given that the conditions are satisfied, the file's status will become "UNUSED".

P5.2 For all f_1 in F ,

$$CLS'(f_1) = \begin{cases} \text{UNDEFINED} & \text{if } f \delta f_1 \wedge C5.1 \wedge C5.2 \\ CLS(f_1) & \text{if } \neg(f \delta f_1 \wedge C5.1 \wedge C5.2) \end{cases}$$

The file destroyed (and any descendants) no longer has any classification defined (assuming of course that the conditions are all satisfied).

P5.3 For all f_1 in F ,

$$Fav'(f_1) = \begin{cases} \text{UNDEFINED} & \text{if } f \delta f_1 \wedge C5.1 \wedge C5.2 \\ Fav(f_1) & \text{if } \neg(f \delta f_1 \wedge C5.1 \wedge C5.2) \end{cases}$$

All other attributes in the affected files will now be undefined also.

P5.4 For all f_1, f_2 in F , $f_1 \delta' f_2$ if and only if $f_1 \delta f_2$
 $\wedge \neg(C5.1 \wedge C5.2 \wedge f \delta f_1)$

For all files that remain in use after the event occurs, the "dominates" relation still holds. Note that if some condition is not satisfied, the event is unsuccessful and no change is made to the files or the dominates relation.

P5.5 For all e_1 in E , f_1 in F , $e_1 \alpha' f_1$ if and only if $e_1 \alpha f_1$
 $\wedge \neg(f \delta f_1 \wedge C5.1 \wedge C5.2)$

Any of the deleted files can no longer be altered. Access remains unchanged if the event is unsuccessful.

P5.6 For all f_1 in F , e_1 in E , $e_1 \nu' f_1$ if and only if $e_1 \nu f_1$
 $\wedge \neg(f \delta f_1 \wedge C5.1 \wedge C5.2)$

If the conditions are met, file f and any of its ancestors may no longer be viewed.

All other properties remain unchanged.

E6 Executor e changes the attribute values of f to V.

Most of the state changes in the system will involve changing the value of some file attribute - usually its contents. In subsequent, more detailed models, this general event will be broken into a number of more specialized operations. Not all attribute values will be contained within the file itself. Some will be located in the directory which immediately dominates the file, and some may be located elsewhere. An executor will have to have the proper capability to access an attribute. As a result, additional restraints will appear in the next chapter. For the present, the classifications of attributes are ignored.

C6.1 $ST(f) = USED$

The file must be in use. Otherwise, it would make no sense to change its attributes.

C6.2 $st(V) = USED$

The file remains in use after the event occurs.

C6.3 $cls(V) = (f)$

The classification of file f also must not change.

P6.3 For all f_1 in F ,

$$Fav'(f) = \begin{cases} V & \text{if } f_1 = f \wedge C6.1 \wedge C6.2 \wedge C6.3 \\ Fav(f_1) & \text{if } \neg(f_1 = f \wedge C6.1 \wedge C6.2 \wedge C6.3) \end{cases}$$

If the conditions are met, file f 's attributes are changed as requested. All other properties are unchanged.

E7: Executor e raises classification of file f to C .

C7. $ST(f) = \text{USED}$

The file must be in use.

C7. For all f_1 in F such that $f \delta f_1$, $CLS(f) \leq C \leq CLS(f_1)$

The new classification must be greater than or equal to the old one. Furthermore, the new classification for f must be less than or equal to that of any file which f dominates. This guarantees that the classifications increase as you get further from the root. In the next chapter, however, we will see that a practical implementation will necessitate an even more constraining condition which will still guarantee increasing classifications in the tree (see operation 10 in Section 6.8).

P7.2 For all f_1 in F

$$CLS'(f_1) = \begin{cases} C & \text{if } f_1 = f \wedge C7.1 \wedge C7.2 \wedge C7.3 \\ CLS(f_1) & \text{if } \neg(f_1 = f \wedge C7.1 \wedge C7.2 \wedge C7.3) \end{cases}$$

File f is raised to the new classification if permitted. All other files are unchanged.

P7.5 For all e_1 in E , f_1 in F $e_1 \alpha' f_1$ if and only if $e_1 \alpha f_1$
 $\wedge [\neg(f \delta f_1 \wedge f = f_1 \wedge C7.1 \wedge C7.2 \wedge C7.3) \text{ OR } C \leq CLR(e_1)]$

The "can-alter" relation may have to be changed to disallow any file whose ancestor is f , if f can no longer be viewed (see next property).

P7.6 For all e_1 in E , f_1 in F , $e_1 \nu' f_1$ if and only if $e_1 \nu f_1$
 $\wedge [\neg(f = f_1 \wedge C7.1 \wedge C7.2 \wedge C7.3) \text{ OR } C \leq CLR(e_1)]$

In order for some executor, e_i to be able to view some file f_j , the information acquisition axiom must be satisfied, that is

$CLS'(f_j) \trianglelefteq CLR'(e_i)$. If the new classification is no longer less than or equal to executor e_i 's classification, then the "can-view" relation must change. Also, if f can no longer be viewed, any file dominated by f can no longer be viewed. However, the conditions guarantee that these dominated files could not have been view-connected before the event. Therefore, only file f need be checked.

All other properties remain unchanged.

E8: Executor e views file f

This is a passive event, in the sense that no visible changes to the state occur. We do however define a "contents" function on the executor-properties function. This might correspond to some register in an actual machine receiving some information.

C8.1 $e \vee f$

The only condition necessary is that executor e is able to view file f .

$$P8.12 \quad Dat'(Ep(e)) = \begin{cases} f. \text{ info} & \text{if C8.1} \\ Dat(Ep(e)) & \text{if } \neg C8.1 \end{cases}$$

If allowed, information moves from file to executor.

We have finished describing the events which involve files and now move on to the other type of repository, namely mailboxes.

E9: Executor $\underline{e}$ becomes "receive-connected" to mailbox $\underline{m}$.

This event allows an executor to receive messages from (and block on) a mailbox.

C9.1 $CLS(m) = CLR(e)$

Both mailbox and executor must have the same security classification since blocking involves both a view and an alter.

P9.9 For all e_1 in E , m_1 in M , $e_1 \beta' m_1$ if and only if $e_1 \beta m_1$
OR $[e_1 = \underline{e} \wedge m_1 = \underline{m} \wedge C9.1]$

If permitted, the specified mailbox is "block-connected".

All other properties are unchanged.

E10: Executor $\underline{e}$ becomes signal-connected to mailbox $\underline{m}$.

C10.1 $CLR(e) \trianglelefteq CLS(m)$

Executor $\underline{e}$ must be allowed to alter $\underline{m}$ and hence it must have a lower security class.

P10.10 For all e_1 in E , m_1 in M , $e_1 \omega' m_1$ if and only if $e_1 \omega m_1$
OR $[e_1 = \underline{e} \wedge m_1 = \underline{m} \wedge C10.1]$

The "can-wake" relation now includes e can-wake m if the condition has been satisfied.

E11: Executor $\underline{e}$ becomes disconnected from mailbox $\underline{m}$.

P11.9 For all e_1 in E , m_1 in M , $e_1 \beta' m_1$ if and only if
 $e_1 \beta m_1 \wedge (e_1 \neq \underline{e} \vee m_1 \neq \underline{m})$

P11.10 For all e_1 in E , m_1 in M , $e_1 \omega' m_1$ if and only if $e_1 \omega m_1$
 $(e_1 \neq \underline{e} \vee m_1 \neq \underline{m})$.

Executor $\underline{e}$ can not be waked via the disconnected mailbox.

E12: Executor e raises classification of mailbox m to C .

C12.1 $M-CLS(m) \leq C$

A mailbox's clearance may only increase.

C12.2 $CLR(e) \leq M-CLS(m)$

Since the classification change is a modification, it must be a writeup.

$$P12.7 \quad M-CLS'(m) = \begin{cases} C & \text{if } C12.1 \wedge C12.2 \wedge m_1 = m \\ M-CLS(m) & \text{if } \neg(C12.1 \wedge C12.2 \wedge m_1 = m) \end{cases}$$

Mailbox m acquires the new classification.

P12.9 For all e_1 in E , m_1 in M , $e_1 \beta' m_1$ if and only if $e_1 \beta m_1$ and $CLR'(e_1) = CLS'(m_1)$

The "can block" relation is changed so that m cannot be blocked upon any longer (if the condition is satisfied) since the classification of mailbox is no longer equal to the clearance of any of the executors which previously blocked on it.

All other properties are unchanged.

E13: Executor e changes the attribute values of m to V .

This event is used to change any other attributes that a mailbox might have.

C13.1 $CLR(e) \leq M-CLS(m)$.

The information dissemination axiom must be obeyed.

C13.2 $cls(V) = M-CLS(m)$

The mailbox's classification must not change.

P13.8 For all m_1 in M ,

$$Mav'(m_1) = \begin{cases} V \text{ if } m_1 = m \wedge C13.1 \wedge C13.2 \\ Mav(m_1) \text{ if } \neg(m_1 = m \wedge C13.1 \wedge C13.2) \end{cases}$$

All other properties are unchanged.

E14: Executor e signals via mailbox m

An actual signal is sent by executor e.

C14.1 $e \omega m$

Executor e can signal via m.

P14.8 For all m_1 in M ,

$$Con'(Mav(m_1)) = \begin{cases} Con(Mav(m_1)) \cup e.message \text{ if } m_1 = m \\ \wedge C14.1 \\ Con(Mav(m_1)) \text{ if } \neg(m_1 = m \wedge C14.1) \end{cases}$$

A new message (in addition to those already there is placed in the mailbox.

All other properties are unchanged.

E15: Executor e blocks on mailbox m.

C15.1 $e \beta m$

Executor e "can-block" on m.

P15.8 For all m_1 in M

$$Con'(Mav(m_1)) = \begin{cases} Con(Mav(m_1)) \wedge \neg m_1.message \text{ if } m_1 = m \wedge \\ C15.1 \\ Con(Mav(m_1)) \text{ if } \neg(m_1 = m \wedge C15.1) \end{cases}$$

If the condition is satisfied, a message is removed from the mailbox.

All other properties are unchanged.

E16: Executor $\underline{e}$ raises its own clearance to $\underline{C}$.

An executor may raise its clearance, however we must make sure that its capabilities are changed to be compatible with the new clearance.

$$C16.1 \quad CLR(e) \leq \underline{C} \leq MAXCLR(e)$$

The change in clearance can only be an increase. In addition, the new clearance must be less than or equal to the user's clearance, i.e. each executor has a maximum clearance.

$$P16.5 \quad \text{For all } e_1 \text{ in } E, f_1 \text{ in } F, e_1 \alpha' f_1 \text{ if and only if } e_1 \alpha f_1 \\ \wedge [C \leq CLS(f_1) \vee \neg C16.1]$$

In order for $\underline{e}$ to continue to be able to alter a file, the file's classification must be greater than or equal to the new clearance.

$$P16.9 \quad \text{For all } e_1 \text{ in } E, m_1 \text{ in } M, e_1 \beta' m_1 \text{ if and only if } e_1 \beta m_1 \\ \wedge \neg(e_1 = e \wedge C16.1)$$

Since executors can block only on mailboxes at the same classification, all presently connected mailboxes must be disconnected, if the event is successful.

$$P16.10 \quad \text{For all } m_1 \text{ in } M, e_1 \text{ in } E, e_1 \omega' m_1 \text{ if and only if } e_1 \omega m_1 \\ \wedge [\neg(e_1 = e \wedge C16.1) \text{ OR } C \leq M-CLS(m_1)]$$

If the condition is satisfied, then a mailbox can remain signal connected only if the executor's new clearance is still lower than the mailbox's classification.

P16.11 For all e_1 in E ,

$$CLR'(e_1) = \begin{cases} C & \text{if } e_1 = \underline{e} \wedge C16.1 \\ CLR(e_1) & \text{if } \neg(e_1 = e \wedge C16.1) \end{cases}$$

The clearance of executor e is changed as specified provided that the condition is met.

All other properties are unchanged.

E17: Executor e changes its properties to P.

This event is used to change properties other than "clearance" and "user".

C17.1 $clr(P)$

The clearance requested must be the same as the present clearance.

C17.2 $user(P) = USER(e)$

The executor still belongs to the same user.

P17.13 For all e_1 in E

$$Ep'(e_1) = \begin{cases} P & \text{if } C17.1 \wedge C17.2 \wedge e_1 = e \\ Ep(e_1) & \text{if } \neg(C17.1 \wedge C17.2 \wedge e_1 = e) \end{cases}$$

If the conditions are satisfied, the executor acquires a new set of properties.

All other properties remain unchanged.

5.3 The Connection to the Previous Specification

Now that the S_3 specification has been presented, our methodology requires that we show that it is an example of the specification presented in chapter 4, and hence that the system is still uncompromisable. Specifically, we must use the axioms of S_3 , (primarily the conditions and properties which accompany the various events) to prove that the axioms of the S_2 specification are satisfied.

Formally this requires us to prove seventeen axioms about each of the seventeen events in the S_3 specification. This would be a rather formidable task except that most of the events only change a few parts of the state of the system and most axioms at S_2 only refer to a few portions of the state. Thus, each event can only effect a few axioms. The proofs can be found in Appendix D.

6. S_4 - COMMANDS AT THE SECURITY PERIMETER

6.1 Introduction

In the previous chapter, we presented a set of prototype operations for the Security System. These prototypes can be thought of as equivalence classes of operations (where all operations belonging to the same class have identical security considerations). The purpose of this chapter is to describe the Security Perimeter - a group of primitives which comprise the virtual machine seen by users (and the operating system). In addition, the mapping between these primitives and the prototypes (of the last chapter) will be shown.

At this point in the specification, it is necessary to introduce further details about the Security System, particularly those details concerning the properties of processes and files. The properties of a process are (for the most part) kept in the Process Segment table, while the properties of a file (its attributes) can be stored within the file system itself.

One file attribute which is of particular interest from the military security point of view is the Access Control List which is used to implement "need-to-know" security restrictions. Since processes can only access attached files, it is only necessary to check the Access Control List attribute when performing the "GET ACCESS" operation (see operation 3 in section 6.8) or the "REMOVE from ACL" operation (see operation 8).

6.2 Processes

In this section we begin to introduce some of the structure of user processes; and in particular we will be interested in the following process attributes.

- 1) the user to whom the processor belongs
- 2) its clearance (security class)
- 3) kinds of access permitted to the files in the file system;
i.e. read, write or both.

The first two pieces of information are stable in the sense that the amount of information is constant; the user always remains the same and only one clearance is associated with a process (though it can be raised). In contrast, access information can change in several ways. First, the type of access to files can change. More significantly, the number of files to which a process has access can change (thereby changing the amount of information about a process). To keep track of its access rights to files in the file system, each process has a table called the process segment table (PST).

6.3 The PST

The Process Segment Table is a local representation of the file system and as such, it contains only those files which are known to the process. (See Biba, et. al [3]) However, the structure of the PST must clearly be related to that of the file system.

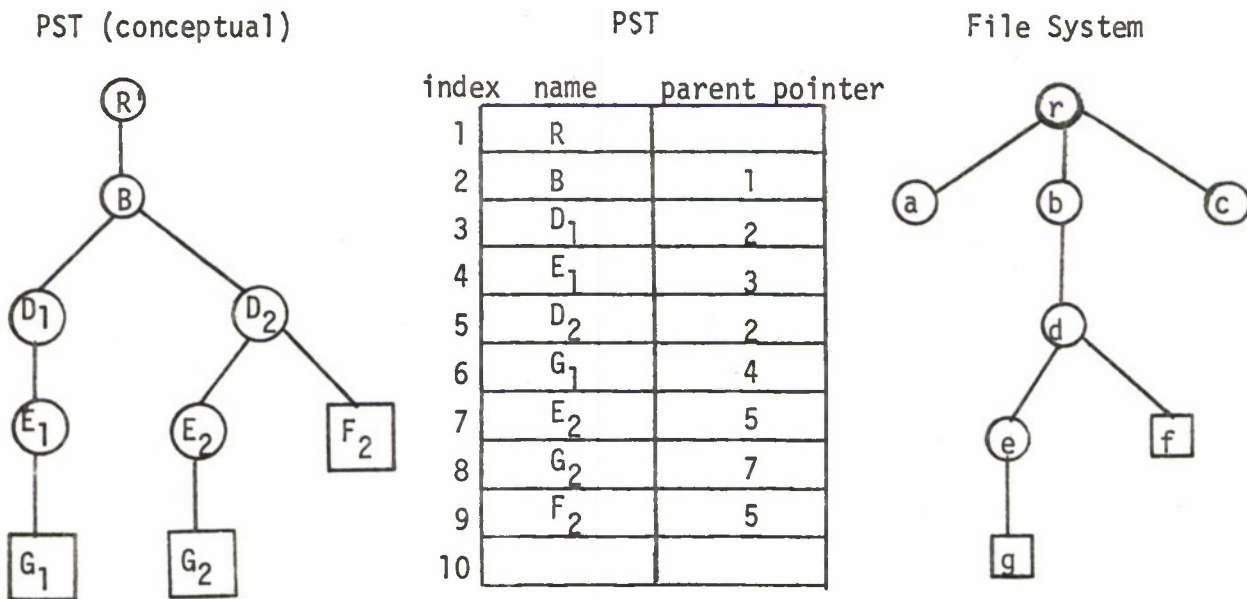


Fig. 6.3.1 PST Representation of the File System

In order to contain sufficient information to represent the structure, each PST entry contains a pointer to its parent (in the PST) and its relative position or entry-number in the parent directory. At first, one might suspect that a process' PST would be a subgraph of the file system; however, in order to reduce the amount of checking which must be done by the security system, we do not insist upon it. We ensure only that there is a homomorphic mapping from the nodes in the process segment table to the nodes of the file system. In other words, for any node in the PST, there is only one node in the File System. There is however, a possibility that two or more nodes in the PST correspond to

the same node in the file system.

In Fig. 6.3.1, E_1 and E_2 map into node e. More will be said about the PST in the section about operations. In particular, we will show how the initiate operation allows more than one PST entry to correspond to a single file. It is also possible that there are files in the file system which are unknown by a process; i.e. there is no entry in the PST corresponding to a node in the File System. In fig. 6.3.1, files a and c are such files.

It seems appropriate now, to give some insight into the implementation of the process segment table. As its name suggests, the PST is a table whose entries represent nodes of the file system. Each node has an implicit name - the index into the table. Information about the tree structure of the PST is contained in links. In particular, each entry contains a pointer to its parent in the PST. Additional information in an entry would include the entry-number (as explained above), a mapping to the file system (see the GET ACCESS operation), an indication of the type of file (directory or segment) and whether the entry is in fact in use. One last piece of information, the childcnt, is used in directory files to keep track of the number of attached offspring it has. This is necessary because in order to remove ACCESS from a file, it must not have any descendents INUSE. Fig. 6.3.2 summarizes the contents of a PST entry.

Parent Link
Ptr. to file system
Type/Inuse
Entryno.
Childcnt

Fig. 6.3.2

6.4 Files

Files were introduced in the S_1 model and in a sense, they are the atomic unit in the information storage system. As in the case of the atom, the file can be broken into smaller pieces, but these smaller pieces (called attributes) have no function by themselves. Just as there are different types of atoms, there are different types of files; in particular there two types - directories and data-segments. Data-segments, represented by the boxes in Fig. 6.3.1, hold the information within the system while the directories provide the means for organizing and finding this information.

Each file consists of a set of attributes and there appear to be two types of attributes (which should be considered separately for specification purposes). The first type consists of attributes which restrict access to a file. This group of attributes consists of CLS (classification), ACL (Access Control List), TYPE and RINGBRACK. The second type includes CURLen, MAXLEN, NAME, DTM (Date & Time Modified) etc., and in a sense they represent physical characteristics of the file. The important difference lies in the fact that modifying arbitrarily one of the access attributes could have disastrous effects; e.g. changing CLS(f) to a lower classification would be equivalent to moving the contents of f to another file with a lower classification which is clearly illegal.

The second type of attribute does not by itself pose a security threat (even if tampered with) however the attributes must be given classifications at which they can be read from and stored into when necessary while satisfying the S_0 axioms. We have intuitively decided

upon the classifications of these attributes (See Ames [1974]) and later when we discuss the primitive commands of the system, we will show that there are no violations of the security axioms.

6.5 Access Attributes

CLS (classification)

This attribute indicates the security level of the file. A file's classification may be altered by three different commands - CREATE FILE, DELETE FILE and RAISE CLASS. Each of these commands must allow the executor to view and alter the file's directory and hence the executor's clearance must be the same as the classification of the file's directory. Since the executor must be able to alter "CLS", it must be at a clearance less than or equal to the classification of the attribute. Thus the classification of CLS could conceivably be either that of the directory or that of the data segment. There is however, one more factor which constrains the attribute's classification: The RAISE CLASS operation must check that the new classification is greater than or equal to the old one. Since it must view CLS, we stipulate that the "CLS" attribute for file f has a classification which equals the classification of d (file f 's parent directory).

ACL (Access Control List)

An ACL is the means of providing discretionary security. By this, we mean that a user A can specify who may have access

privileges to any of his files. For a user to access such a file, he must have both the proper clearance and A's permission. Since it is essential that a user find out whether he may access a file f before actually accessing the file, we stipulate that f's ACL is associated with (and at the same classification as) f's parent directory.

TYPE

Although it is not obvious, this attribute also serves to restrict access to a file. A file's type may be: DIRECTORY, DATASEGMENT or UNUSED. If a file's TYPE is DIRECTORY, or UNUSED the following special restrictions are placed on access privileges: A directory can only be accessed by system commands (for integrity reasons), while an unused file just doesn't exist and cannot be accessed at all. A file's type must be viewed if its classification is to be changed. Since it must be viewed and altered by operations working at the directory's classification, TYPE must also be at that classification.

RINGBRACKETS

This attribute is used as an integrity mechanism in Multics. Ringbrackets provide the means for restricting the access of directories to ring-0 (privileged system) routines and facilitate a distributed operating system whereby another user's interrupts may be handled without switching processes. RINGBRACKETS must be viewed before access to a file is permitted and is necessarily at the directory's classification.

6.6 Characteristic Attributes

AUTHOR

There is no apparent necessity for maintaining this attribute which designates the user who originally created the file. If desired, it should be located in the directory since it will only be altered upon creation and deletion.

BITCNT (bit count)

This attribute specifies (with single bit granularity) how much of a data-segment is being used. (It is not applicable to directories). It is useful for printing files etc. but is ignored by the security system. (It is somewhat analogous to the unofficial clock at a football game). The attribute must be at the same classification as the data-segment so that the user can view and alter it while working on the file. We use the terminology "logically located in the data-segment" to express the notion that conceptually, the attribute is located with (and at the same classification as) the contents of the file. However, it is expected that in the actual implementation, this attribute will physically reside in the directory. Still, the use of system routines to interperatively access items in a directory allows the system to "make believe" the attribute is really in the data-segment; that is it will obey the security restrictions as if it were at the classification of the data-segment.

CURLEN (current length)

This attribute is an integrity mechanism (though not the only line of defense) to prevent exceeding a data-segment's boundary and illegally accessing another file. See Schroeder and Saltzer [11] for a discussion of how Multics' virtual memory hardware also performs this function. Since the current length changes as the file is altered, it must be logically-located in the file itself. (Refer to BITCNT for a discussion of "logical-location"). Note: this differs from the present Multics implementation.

CONTENTS

The contents attribute (only for a data-segment) is the information which is kept there. Unlike directories, a data segment may be filled arbitrarily and directly by users with appropriate access privileges and likewise may be directly viewed by users. The contents of a file, by definition, reside in the file and at the file's classification.

COPYSWITCH

This attribute (used only for data-segments) assures each user his own copy of the file. This is necessary for segments containing non-reentrant code. The user need not know about this attribute, however, for security purposes, it should be located in the directory so that it may be viewed during the "GET ACCESS" command which operates at the directory's classification.

DTD (Date/Time Dumped)

This attribute is used by the file system's backup mechanism. All files are saved regularly so that restoration can be achieved in the event of a system failure. Presumably, some process at the file's clearance must view the file, copy it elsewhere and set DTD. Since the DTD attribute must be altered, it should be logically-located with the file.

DTEM (Date and Time Entry Modified)

This attribute is presently used in Multics to indicate when any of a file's attributes (besides contents of a data-segment) was last changed. Since some attributes are logically-located with the file itself, while others are logically-located with the directory, the DTEM would have to be logically-located with the file. Alternatively, the DTEM could be separated into two attributes - DTFEM and DTDEM. The first would reflect changes to attributes logically-located with the file while the second would represent those of the directory. Each would be logically-located with the attributes it monitors.

DTM (Date/Time Modified)

This is similar to DTEM (above) but is modified when the contents (of a data-segment) are altered. It too must be logically located with the file.

DTU (Date/Time Used)

In present Multics, DTU keeps track of when a file has been accessed (in any manner). It is useful for determining the secondary storage medium on which a file should be kept. For example, a file accessed in the last few seconds should probably be swapped to bulk core or fast drum since it is likely to be used again, while a file that hasn't been referenced in several months might be moved from disk to tape in order to save disk space. Despite its usefulness, DTU poses a security threat. This is because a file may be viewed from any clearance which is greater than the classification of the file. Since the DTU attribute is altered as a result of this observation, it must be at a higher classification than the agent which viewed the file. To satisfy the most general case, it cannot be stored in the directory or file and in general, it cannot be viewed by the owner of the file. These serious restrictions indicate that this attribute should be maintained within the security system and hidden from the user. For alternative schemes, see Biba, et. al [3].

IACL (Initial Access Control List)

Specifying an ACL each time a file is created could become annoying, especially if the ACL is complex. Multics has given the user the capability of specifying a default ACL known as the Initial Access Control List or IACL. An IACL is associated only with directory files and stipulates that all files created in it (without an ACL specified) will use

a copy of the IACL as their ACL. Since in Multics, the IACL can be changed by anyone with modify access to the (directory) file with which it is associated, the IACL must be logically-located in the (directory) itself.

NAMES

To paraphrase a famous poem, "A file with any other name would still contain the same information". One of the features of Multics is that a file can have any number of names by which users can refer to it. In order to simplify the security system a file is accessed as an index into the Process Segment Table (see sec. 6.3). However for convenience, the operating system will provide users with reference names for the file. Since the NAMES attribute must be viewed by the operating system's "Delete" command, and altered by the system's "Create" command (both of which occur at the directory's classification), the NAMES attribute must be logically-located with the directory.

QUOTA

INFQCNT (Inferior Quota Count)
SPUSED (Space Used)
TACCSW (Terminal Account Switch)

These four attributes are involved with the file system quota mechanism. As the name implies, quotas are used to limit secondary storage allocation. Only directories may have quotas in present Multics, and not all directories need quotas. A file attribute called the Terminal Account Switch

(TACCSW) determines whether the directory may have a quota. When more storage space is needed by a file, this space is charged against the quota of the first directory encountered (going up the tree towards the root) which has its TACCSW set on. (As will be explained later in more detail, this presents a security problem). The "Space Used" (SPUSED) attribute indicates the amount of storage charged against the quota of a directory. The difference between QUOTA and SPUSED gives the amount of storage space that may still be assigned.

A security problem occurs if a user is writing in a data-segment and additional space becomes necessary. The process must be capable of altering the file and of viewing and altering the QUOTA, SPUSED and TACCSW attributes of some superior directory. Clearly the process would have to be at a clearance equal to the classification of the directory file which contains these attributes, and unless blind writing of the data-segment were being done, the classification of the segment would have to be the same. In other words all data-segments would have to have their parent's classification. Alternatively, data segments could be given their own quota which might be embodied in the MAXLEN attribute. SPUSED would then be taken care of by CURLEN. Clearly the space usage would have to be anticipated and MAXLEN set ahead of time. Also, in the directory structure, TACCSW would have to be set at each direc-

tory file which had a classification different from its parent's.

One restriction still necessary is that quota may only move down the tree (away from the root) except in the case of DESTROY SUBTREE (see sec. 6.8). Quota may only be sent down the tree - never requested and it may only be taken back in its entirety. Thus no information is passed downward in the form of a quota request or encoded in a quota return.

RECUSED (Records Used)

This attribute indicates how much storage space is used by an individual file. Since RECUSED must be modified when the file is modified, it must be logically-located with the file. Thus, RECUSED may only be observed if the file can also be observed; it may not be observed from the directory containing the file as is the case with the present Multics implementation.

SAFSW (Safety Switch)

In present Multics, the SAFSW attribute is used as a lock to prevent a file from being deleted. This attribute is useless since the "Delete" command will have to allow blind deletes. (See Delete, sec. 6.8)

DID (Device Identifier)

This attribute tells what type of storage device the file is stored in. It would have to be at the classification of the file itself so that when a file is moved from one device to another the DID attribute can be observed and modified. Alternatively, this attribute could be hidden from the user.

UID (Unique Identifier)

The purpose of the UID in present Multics is to assure that any file (possibly accessed by more than one name) appears in core only once. Since a file's UID may not change (it is view-only) it can be logically located with the directory.

6.7 Process Attributes

Associated with any process will be three basic attributes: Process Clearance (PCLR), Process Segment Table (PST) and Process User (PUSER). These attributes will be described here.

PCLR (Process Clearance)

This attribute gives the clearance of a process (which must be at a security level attainable by the owner of the process). Furthermore, restrictions must be made on how a process' clearance can change. We will stipulate that it can be changed only by the "CHANGE CLEARANCE" operation (see sec. 6.8) and the clearance can only increase.

PUSER (Process User)

A process can belong to exactly one user and that user will remain the same throughout the life of the process. Hence the PUSER attribute may not be altered.

Though the PST is an attribute of a process, it is both unusual and complex. As mentioned before, it is the process' "picture" of the file system. Each known file will have at least one PST entry, although some redundancy is possible. The subentries of the PST will be summarized here.

ALPAR (Alleged Parent)

When an entry is created in the PST, a pointer indicating the parent is created. Since no information about this entry (or the entry pointed to) has been verified with the information in the file system, we use the term "alleged parent". As long as the PST is not attached to some file, information in the entry may be invalid.

CHILDCNT (Childcount)

This attribute reflects the number of entries which have been initiated directly inferior to this entry. It is useful in making sure that a directory has no offspring still attached to the PST. This is necessary since there are no downward pointers in the PST, and CHILDCNT is relied upon to determine whether a file can be detached.

ATTACHED

To be of any use, an entry in the PST must be associated with or ATTACHED to some file in the file system. The ATTACHED attribute is the boolean which indicates whether the GET ACCESS operation has in fact taken place. When ATTACHED is true, the information in the PST can be regarded as being accurate.

MAP

This is the internal pointer in the PST entry which provides a handle on the file system. It can be used only after the file has been attached and is set up during the

GET ACCESS operation.

PENTRYNO (PST Entry Number)

The file system can be thought of as an ordered tree (see Knuth vol. I, sec. 2.3.4.2) since the order of the branches is important. As a result, any file can be uniquely described by two pieces of information: 1) its parent and 2) the particular branch or Entry Number within the parent directory. Given that all processes are attached to the root, all desired files can be described by working down the tree. In general, the ALPAR and PENTRYNO attributes will provide enough information to access a file, since PENTRYNO corresponds to the Entry Number of the desired file within its parent directory.

6.8 Primitive Operations

We are now ready to discuss a set of primitive operations for the system. Our goal is to provide a minimal set of primitives which can be certified and which will support a Multics-like system.

Basically, each of the operations manipulates one or more of the previously described attributes. By placing appropriate restrictions on critical attributes (those listed as "Access Attributes") and making sure that the process has appropriate access permission to all attributes used to carry out a primitive, we feel that a primitive can be certified.

Before jumping into the descriptions of these primitives, a discussion of our notation might be helpful. We have chosen script letters to represent functions, e.g. *alpar*, *type*, *Al*. They return the value of the attribute with the same name. Functions beginning with a

capital letter return several values, e.g. *VL* (view-list) which returns a list of users. Conversely those beginning with a small letter return a single value such as a file's type or classification. Composite functions will often be used to simplify notation and will be represented by capital script letters. For example in the GET ACCESS primitive, while attempting to attach a PST entry to some file, the TYPE of the file must be ascertained by the composite function:

$$p\text{-type}'(F) = \text{type} [\text{branch} (\text{pentryno}(F), \text{map} (\text{alpar}(F)))]$$

using shorthand, we will write:

$$p\text{-type}'(F) = \text{TYPE}(f)$$

A function's arguments will typically consist of file names or PST entries. We will use capital arabic letters for PST entry names though the reader should keep in mind that it is actually an index into a table. Files will be represented by small arabic letters though in actuality, files can only be referred to by a pointer (from the PST) to the file's parent directory and an entry number within the directory. For our descriptions, the letters in the PST will match those in the file system (see fig. 6.3.1) however there is really no relationship between them.

The primitive operations will be described in terms of conditions and properties. A condition will have a value of TRUE or FALSE, while the properties will represent the end result of the operation. Note that a property may depend on various conditions being true.

Each of the properties has been given a number for organizational purposes. The descriptions in the next section list these properties as $P_{x.n}$ where x stands for the operation number and n represents the

property number. Table 6.8.1 summarizes the properties and their numbers:

Px.1	<i>cls</i>	(classification)
Px.2	<i>mayview, mayalter</i>	(These correspond to discretionary view and alter permission. See ADD ACL, and the GETACCESS operations, sec. 6.8.)
Px.3	<i>type</i>	
Px.4	<i>Ring</i>	(ring brackets)
Px.5	<i>d-char</i>	(characteristics or attributes logically located in the directory)
Px.6	<i>l-char</i>	(attributes logically located <u>locally</u> or with the file).
Px.7	<i>p-user</i>	(process' user)
Px.8	<i>p-clr</i>	(process' clearance)
Px.9	<i>p-type</i>	(process' type)
Px.10	<i>alpar</i>	(alleged parent)
Px.11	<i>childcnt</i>	(childcount)
Px.12	<i>attached</i>	
Px.13	<i>map</i>	
Px.14	<i>p-entyno</i>	(entry number)

Table 6.8.1

The conditions for each operation are also assigned numbers which begin with the letter "C", however, the number has no implicit meaning.

One additional piece of information is given to indicate the access privileges needed to test a condition or satisfy a property. It is located near the right margin in parenthesis and has the form (p μ PST) or (p θ f) etc. where θ and μ stand for observe and modify respectively while "PST" or "f" indicate the repository being accessed. The "p" indicates that the process is making the access. Also, "No

access required" is indicated by a hyphen, i.e. (-).

1.) INITIATE PST entry F as entry number eno in D with type I

The INITIATE operation is used to place an entry in the Process Segment Table. In keeping with the Multics philosophy that a file doesn't become attached until it is accessed, INITIATE does no more than build the entry. This information is not considered valid until the GET ACCESS operation has attached the file.

C1.1 $p\text{-type}(F) = \text{UNUSED}$ (p $\in$ PST)

First, the PST entry must not be in use since information which is presently valid would be destroyed.

C1.2 $p\text{-type}(D) = \text{DIRECTORY}$ (p $\in$ PST)

Since we are initiating a file in D, we must be certain that D is expected to be a directory. If not, the modification of CHILDCNT (see P1.11) would be invalid.

P1.1-6 There are no changes made to the file system. (-)

P1.7 $\forall p_1 \in P \ p\text{-user}'(p_1) = p\text{-user}(p_1)$ (-)

The USER who owns the process cannot be changed. One assumption which has been made is that a PST is created with P-USER correctly established.

P1.8 $\forall p_1 \in P \ p\text{-clr}'(p_1) = p\text{-clr}(p_1)$ (-)

The clearance of the process remains the same. Recall that only one operation, CHANGE CLEARANCE, is able to change a process' clearance.

Again, it is assumed that a PST's P-CLR attribute is properly initiated.

P1.10 $\forall F_1 \in \text{PST} \ alpar'(F_1) = \begin{cases} D & \text{if } (F_1 = F) \wedge C1.1 \wedge C1.2 \\ alpar(F_1) & \text{otherwise} \end{cases}$ (p $\in$ PST)

The process sets up a pointer to the PST entry it thinks is the desired parent. (see explanation of PST sec. 6.3)

$$P1.11 \quad \forall F_1 \in \text{PST} \quad \text{childcnt}'(F_1) = \begin{cases} 0 & \text{if } (F_1=F) \wedge C1.1 \wedge C1.2 \\ \text{childcnt}(F_1) & \text{otherwise } (p \in \text{PST}) \end{cases}$$

The CHILDCNT attribute must be initialized to zero reflecting the fact that no files have been initiated inferior to $\underline{F}$.

$$\text{childcnt}'(D) = \begin{cases} \text{childcnt}(D) + 1 & \text{if } (F_1=F) \wedge C1.1 \wedge C1.2 \\ \text{childcnt}(D) & \text{otherwise } (p \in \text{PST}) \end{cases}$$

The CHILDCNT attribute of D must be altered to reflect the fact that a new file has been INITIATED below it.

$$P1.12 \quad \forall F_1 \in \text{PST} \quad \text{attached}'(F_1) = \begin{cases} \text{FALSE} & \text{if } (F_1=F) \wedge C1.1 \wedge C1.2 \\ \text{attached}(F_1) & \text{otherwise } (p \in \text{PST}) \end{cases}$$

The ATTACHED attribute is made TRUE by the GETACCESS operation and must be FALSE until that time.

$$P1.13 \quad \forall F_1 \in \text{PST} \quad \text{map}'(F_1) = \begin{cases} \text{NULL} & \text{if } (F_1=F) \wedge C1.1 \wedge C1.2 \quad (p \in \text{PST}) \\ \text{map}(F_1) & \text{otherwise} \end{cases}$$

The MAP attribute is the process' handle on the file, however it is not considered valid until the file has been attached.

$$P1.14 \quad \forall F_1 \in \text{PST} \quad p\text{-entrynd}'(F_1) = \begin{cases} \text{eno} & \text{if } (F_1=F) \wedge C1.1 \wedge C1.2 \\ p\text{-entryno}(F_1) & \text{otherwise } (p \in \text{PST}) \end{cases}$$

The Process Entry Number is set to specify the file within the directory (in the file system) which is desired.

$$P1.15 \quad \forall F_1 \in \text{PST}, \forall n \in \mathbb{N}$$

$$\text{branch}'(\text{map}(F_1), n) = \text{branch}(\text{map}(F_1), n) \quad (-)$$

This property states that the structure of the file system is unchanged or, more specifically, that each directory file keeps the same files as offspring.

2.) TERMINATE PST entry F.

This operation can be thought of as the opposite of INITIATE. Its purpose is to remove an entry from the PST. However, several checks must be made before the operation occurs.

$$C2.1 \quad childcnt(F) = 0 \quad (p \in PST)$$

The file must have no files initiated beneath it. i.e. it must be either a directory with no offspring initiated or a datasegment (which obeys this condition by definition).

P2.1-6 No file attributes are changed.

$$P2.7 \quad \forall p_1 \in p\text{-user}'(p_1) = p\text{-user}(p_1) \quad (-)$$

$$P2.8 \quad p\text{-clr}'(p) = p\text{-clr}(p) \quad (-)$$

$$P2.9 \quad \forall F_1 \in PST \quad p\text{-type}'(F_1) = \begin{cases} \text{UNUSED} & \text{if } C2.1 \wedge (F_1 = F) \\ p\text{-type}(F_1) & \text{otherwise} \end{cases} \quad (p \in PST)$$

The purpose of this operation is to make an entry available (which amounts to setting the P-TYPE attribute to UNUSED).

$$P2.10 \quad \forall F_1 \in PST \quad alpar'(F_1) = \begin{cases} \text{UNDEFINED} & \text{if } (F_1 = F) \wedge C2.1 \\ alpar(F_1) & \text{otherwise} \end{cases} \quad (p \in PST)$$

$$P2.11 \quad \forall F_1 \in PST \quad childcnt'(F_1) = \begin{cases} 0 & \text{if } (F_1 = F) \wedge C2.1 \\ childcnt(F_1) & \text{otherwise} \end{cases} \quad (p \in PST)$$

$$P2.12 \quad \forall F_1 \in PST \quad attached'(F_1) = \begin{cases} \text{FALSE} & \text{if } (F_1 = F) \wedge C2.1 \\ attached(F_1) & \text{otherwise} \end{cases} \quad (p \in PST)$$

$$P2.13 \quad \forall F_1 \in PST \quad map'(F_1) = \begin{cases} \text{NULL} & \text{if } (F_1 = F) \wedge C2.1 \\ map(F_1) & \text{otherwise} \end{cases} \quad (p \in PST)$$

$$P2.14 \quad \forall F_1 \in PST \quad p\text{-entryno}'(F_1) = \begin{cases} \text{NULL} & \text{if } (F_1 = F) \wedge C2.1 \\ p\text{-entryno}(F_1) & \text{otherwise} \end{cases} \quad (p \in PST)$$

These attributes are all set to some "safe" value. Although they could just as well be "Don't Care" since INITIATE initializes them, as a precaution, they should be set to NULL.

$$\begin{aligned} \text{P2.15} \quad \forall F_1 \in \text{PST}, \forall n \in \mathbb{N} \quad & \text{branch}'(\text{map}(F_1)n) & (-) \\ & = \text{branch}(\text{map}(F_1), n) \end{aligned}$$

The file system structure is not changed.

3.) GET ACCESS a for F

The Purpose of this operation is to give the process a handle on the file system. This involves checking all the information in the PST for correctness and setting the ATTACHED and MAP attributes to appropriate values. Actually, the ATTACHED attribute consists of three parts; VIEW-ATTACHED, ALTER-ATTACHED, and ATTACHED. The last of these indicates that the file is attached in some manner, while the first two describe the process' capabilities for a given file. When the GET ACCESS operator has successfully occurred, we know that mandatory security (classifications), discretionary security (ACLs - *mayview*, *mayalter*) and ringbrackets have been checked. In appendix E, we show that the value of the "ATTACHED" attributes remains consistent with the security axioms; in otherwords we can assume that we do indeed have the access capabilities specified by these attributes.

The argument a specifies the privileges desired by the process for file F. Formally $\underline{a} \subseteq \{\text{view-desire}, \text{alter-desire}\}$. Note that a has been specified during the initiate operation and is an implied argument.

$$\text{C3.1} \quad \text{view-attached}(\text{alpar}(F)) = \text{TRUE} \quad (p \in \text{PST})$$

For this operation to take place, the parent directory for F (in the file system) must be viewed. Note that this is equivalent to:

$$\text{view-attached}(D) = \text{TRUE}$$

where D is F's alleged parent. This shortened form of notation will be used throughout this section although D is not specified and must be ascertained during the operation.

$$\text{C3.2} \quad \text{p-type}(D) = \text{DIRECTORY} \quad (p \in \text{PST})$$

Since D is allegedly F's parent, we would like to make sure that D is a directory.

$$C3.3 \quad TYPE(f) \in \{DATASEGMENT, DIRECTORY\} \quad (p \neq d)$$

This condition is necessary to make sure that there is something to which this PST entry can be attached.

$$C3.4 \quad [ring \leq accessbound(f) \vee \neg a.alter] \wedge \\ [ring \leq callbound(f) \vee \neg a.view]$$

The ring bracket accessing requirements must be met. See Organic, [9] for details about ring brackets.

$$P3.1-6 \quad \text{All file system attributes remain unchanged.} \quad (-)$$

$$P3.7,8 \quad p\text{-user}, p\text{-clr} \text{ remain constant.} \quad (-)$$

$$P3.9 \quad \forall F_1 \in PST \quad p\text{-type}(F_1) = \begin{cases} TYPE(f) & \text{if } (F_1 = F) \wedge C3.1 \wedge \dots \wedge C3.4 \\ p\text{-type}(F_1) & \text{otherwise } (p \in PST, p \neq d) \end{cases}$$

The P-TYPE attribute (in the PST entry) is set to the TYPE attribute of the file in the file system.

$$P3.10 \quad \forall F_1 \in PST \quad alpar'(F_1) = alpar(F_1)$$

Since this function was used to locate $\underline{f}$ in the file system, it is correct by definition in the sense that $\underline{f}$ is some specific offspring of $\underline{d}$, $\underline{F}$ is the corresponding offspring of $\underline{D}$ and $\underline{D}$ corresponds to (or maps into) $\underline{d}$.

$$P3.11 \quad \forall F_1 \in PST \quad childcnt'(F_1) = \begin{cases} 0 & \text{if } (F_1 = F) \wedge C3.1 \wedge \dots \wedge C3.4 \\ childcnt(F_1) & \text{otherwise } (p \in PST) \end{cases}$$

A newly attached file cannot yet have any offspring attached. Note that with proper restrictions on other operations (namely INITIATE, TERMINATE and REMOVE ACCESS) we could have:

$$\forall F_1 \in PST \quad childcnt'(F_1) = childcnt(F_1) \quad (-)$$

$$P3.12 \quad view\text{-attached}'(F_1) = (a.view \wedge [VIEW\text{-ACCESS}(f,u)] \wedge \\ (F_1 = F) \wedge C3.1 \wedge \dots \wedge C3.4 \wedge \quad (p \in PST) \\ CLS(f_1) \leq p\text{-clr}(p)] \vee view\text{-attached}(F_1) \quad (p \neq d, PST)$$

This property will give a process the "can view" capability if permitted and desired. The term, "a.view" is a boolean used to indicate that "view desire" is true. Besides satisfying C3.1-3, user u must have view permission in file f's Access Control List (denoted by the function *VIEW-ACCESS*), and the "information acquisition" property concerning Security Classifications must hold. (See ACL attributes, Section 6.5).

$$\begin{aligned} \forall F_1 \in \text{PST } \text{alter-attached}'(F_1) = & (a.\text{alter} \wedge (p \neq d, \text{PST}) \\ & \wedge (F_1 = F) \wedge C3.1 \wedge C3.2 \wedge C3.3 \wedge C3.4 \wedge (p \neq \text{PST}) \\ & [\text{ALTER-ACCESS}(f, u)] \wedge [p\text{-clr}(p) \leq \text{CLS}(f_1)]) \\ & \vee \text{alter-attached}(F_1) \end{aligned}$$

This property is similar to the previous one, except that it gives "can alter" capability to the PST. Again ACL and CLS attributes must be checked.

$$\forall F_1 \in \text{PST } \text{attached}'(F_1) = \begin{cases} \text{TRUE if } (F_1 = F) \wedge C3.1 \wedge C3.2 \wedge C3.3 \wedge C3.4 \\ \text{attached}(F_1) \text{ otherwise} \end{cases} \quad (p \neq \text{PST})$$

Note that the case where a file is attached with no access privileges is taken care of.

$$\text{P3.13 } \forall F_1 \in \text{PST}, \text{map}'(F_1) = \begin{cases} \text{branch}[\text{map}(\text{alpar}(F_1), p\text{-entryno}(F_1))] \\ \quad \text{if } (F_1 = F) \wedge C3.1 \wedge \dots \wedge C3.4 \\ \quad \quad \quad (p \neq c, \text{PST}) \\ \text{map}(F_1) \text{ otherwise} \end{cases} \quad (p \neq \text{PST})$$

The purpose of this property is to give the PST a handle of some sort on the actual file. (In the implementation, this would be a secondary storage address or pointer). Note that both the entry number and alleged parent which were specified in the INITIATE operation were used to accomplish this operation, and there is an automatic correspondence (as described in sec. 5.3) between the PST and the File System since the attributes P-ENTRYNO and ALPAR cannot be tampered with.

$$P3.14 \quad \forall F_1 \in PST, p\text{-entryno}'(F_1) = p\text{-entryno}(F_1) \quad (-)$$

The P-ENTRYNO attribute must not be changed.

$$P3.15 \quad \forall F_1 \in PST, \forall n \in \mathbb{N}, \text{branch}'(\text{map}(F_1), n) = \text{branch}(\text{map}(F_1), n)$$

There is no change to the file system.

4.) REMOVE ACCESS for F

The purpose of this operation is to invalidate an entry in the PST. It might be called upon if an error were made, the process terminated, space in the PST were scarce or there were some cost involved maintaining an (unneeded) PST entry. In general, we would like this operation to place us in a state which followed the INITIATE operation but preceeded the GET ACCESS primitive for file F.

C4.1 $attached(f) = TRUE$ (p $\in$ PST)

We must check that F is indeed a valid entry. At first one might conjecture that no checks are necessary since if it were an invalid (un-attached) entry before the operation, it would still be invalid afterwards. However, as indicated by the next condition, we want to keep the CHILDCNT attribute accurate (for both F and F's alleged parent) and thus we must do checking.

C4.2 $childcnt(F) = 0$ (p $\in$ PST)

As noted previously, a file can be detached from a process only if none of its offspring is attached.

P4.1-8 unchanged

P4.9 $\forall F_1 \in PST, p-type'(F_1) = p-type(F_1)$

Since a TYPE may be specified by the INITIATE operation and we know that it is presently correct, we can leave it as is.

P4.10 $\forall F_1 \in PST, alpar'(F_1) = alpar(F_1)$

This also is specified by INITIATE.

P4.12 $\forall F_1 \in PST, view-attached'(F_1) = \begin{cases} FALSE \text{ if } (F_1 = F) & (p \in PST) \\ \wedge C4.1 \wedge C4.2 \\ view-attached(F_1) & \text{otherwise} \end{cases}$

$$\forall F_1 \in \text{PST} \quad \begin{aligned} \text{alter-attached}'(F_1) &= \begin{cases} \text{FALSE} & \text{if } (F_1 = F) \wedge C4.1 \wedge C4.2 \\ \text{alter-attached}(F_1) & \text{otherwise} \end{cases} \\ \text{attached}'(F_1) &= \begin{cases} \text{FALSE} & \text{if } (F_1 = F) \wedge C4.1 \wedge C4.2 \\ \text{attached}(F_1) & \text{otherwise} \end{cases} \end{aligned}$$

This property represents the operation's purpose - to remove access privileges from (i.e. detach) the file.

$$P4.13 \quad \forall F_1 \in \text{PST}, \text{map}'(F_1) = \begin{cases} \text{NULL} & \text{if } (F_1 = F) \wedge C4.1 \wedge C4.2 \\ \text{map}(F_1) & \text{otherwise} \end{cases}$$

The handle to the file system is no longer valid.

$$P4.14 \quad \forall F_1 \in \text{PST} \quad p\text{-entryno}'(F_1) = p\text{-entryno}(F_1)$$

Since this is specified by INITIATE, it must remain unchanged.

$$P4.15 \quad \text{The file system structure is unchanged.}$$

5.) CREATE file f as entry eno in directory D with attributes V

This operation causes a new file to come into existence in the file system. The new file can have almost an arbitrary set of attributes (with the exception of CLS which must be greater than or equal to $cls(d)$).

$$C5.1 \quad \text{view-attached}(D) = \text{TRUE} \quad (p \in \text{PST})$$

Some attributes in d must be examined (e.g. $\text{type}(f)$). Hence, view privilege is required.

$$\text{alter-attached}(D) = \text{TRUE} \quad (p \in \text{PST})$$

The operation of creating a file f involves setting of attributes which are kept in d.

$$C5.2 \quad p\text{-type}(D) = \text{DIRECTORY} \quad (p \in \text{PST})$$

A file can be created only in a directory.

$$C5.3 \quad TYPE(f) = \text{UNUSED} \quad (p \neq d)$$

Only one file can occupy any given entry in a directory. This condition precludes the existence of such a file before this operation is carried out.

$$C5.4 \quad p\text{-cls}(D) \leq \text{cls}(V) \quad (p \neq \text{PST})$$

Since the CLS attribute of a file must be greater than or equal to its parent's classification, this check must be done.

$$C5.5 \quad \text{type}(V) \in \{\text{DATASEGMENT}, \text{DIRECTORY}\}$$

A valid type must be requested.

$$C5.6 \quad \text{ring}(V) \geq \text{ring}(p)$$

In present Multics, a process may not create a file with a ring number less than its present ring number. This is a restriction to prevent a Trojan Horse (see Anderson [10]).

$$P5.1 \quad \forall f_1 \in F, \text{CLS}'(f_1) = \begin{cases} \text{cls}(V) & \text{if } f_1 = \text{branch}(d, \text{eno}) \wedge C5.1 \wedge \dots \wedge C5.6 \\ \text{CLS}(f_1) & \text{otherwise} \end{cases} \quad (p \neq d)$$

$$P5.2 \quad \forall f_1 \in F, \text{ACL}'(f_1) = \begin{cases} \text{acl}(V) & \text{if } f_1 = \text{branch}(d, \text{eno}) \wedge C5.1 \wedge \dots \wedge C5.6 \\ \text{ACL}(f_1) & \text{otherwise} \end{cases} \quad (p \neq d)$$

$$P5.3 \quad \forall f_1 \in F, \text{TYPE}'(f_1) = \begin{cases} \text{type}(V) & \text{if } f_1 = \text{branch}(d, \text{eno}) \wedge C5.1 \wedge \dots \wedge C5.6 \\ \text{TYPE}(f_1) & \text{otherwise} \end{cases} \quad (p \neq d)$$

$$P5.4 \quad \forall f_1 \in F, \text{RINGB}'(f_1) = \begin{cases} \text{ring}(V) & \text{if } f_1 = \text{branch}(d, \text{eno}) \wedge C5.1 \wedge \dots \wedge C5.6 \\ \text{RINGB}(f_1) & \text{otherwise} \end{cases} \quad (p \neq d)$$

$$P5.5 \quad \forall f_1 \in F, \text{D-CHAR}'(f_1) = \begin{cases} \text{d-char}(V) & \text{if } f_1 = \text{branch}(d, \text{eno}) \\ & \wedge C5.1 \wedge \dots \wedge C5.6 \\ \text{D-CHAR}(f_1) & \text{otherwise} \end{cases} \quad (p \neq d)$$

$$P5.6 \quad \forall f_1 \in F, \text{L-CHAR}'(f_1) = \begin{cases} \text{l-char}(V) & \text{if } f_1 = \text{branch}(d, \text{eno}) \wedge C5.1 \wedge \dots \wedge C5.6 \\ \text{L-CHAR}(f_1) & \text{otherwise} \end{cases}$$

The attributes for $\underline{f}$ are set if the conditions are met.

P5.7-14 $p\text{-user}$, $p\text{-cls}$ and the PST attributes remain untouched
since the file has not yet been initiated.

$$\text{P5.15} \quad \forall d_1 \in F, \forall n \in \mathbb{N}, \text{branch}'(d_1, n) = \begin{cases} f \text{ if } d_1 = \text{map}(D) \wedge n = \text{eno} \\ \quad \wedge C5.1 \wedge \dots \wedge C5.6 \\ \text{branch}(d_1, n) \text{ otherwise} \end{cases}$$

6.) DESTROY SUBTREE whose root is F

This operation is used to delete files from the file system. A trivial case occurs when F is a data segment (the subtree consists of a single node). Due to restrictions involving ring brackets and quota, this is in general the only permissible way to delete files.

In the present implementation of Multics, a user may not delete a file with a lower ring bracket. However suppose (as illustrated in fig. 6.8.1) there is a subtree of files which increase in classification and decrease in ring number. Obviously, the process cannot view the ring attribute for file h. Furthermore, file g, which dominates h could be destroyed. Thus we have the undesirable possibility of a loose subtree.

Another problem is the return of quota from a more classified directory to one with lower classification. As noted in sec. 6.6, the only way to prevent flow of information is to return the full quota allotment.

These two reasons force the DESTROY SUBTREE operation to ignore ring brackets. Although there could be more restricted forms of this operation under favorable circumstances (e.g. a subtree at a single classification with high enough ring brackets) the operating system could itself enforce the restrictions by walking the subtree and hence the security system need not be complicated further.

Although this operation is unaesthetic, without it, a file system with more than one classification would be unworkable. Various other alternatives were explored by people at the MITRE Corp. (See Biba, et.al, [3])

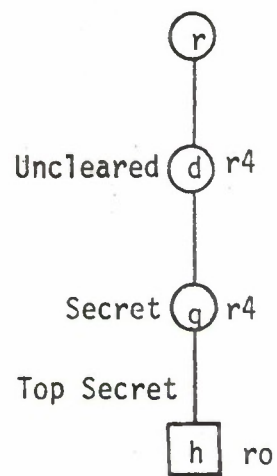


Fig. 6.8.1 - Ring brackets could not be checked when an uncleared user tried to DELETE SUBTREE with root d.

- C6.1 $view-attached(D) = TRUE$ $(p \in PST)$
 $alter-attached(D) = TRUE$ $(p \in PST)$

The directory file which contains the root of the subtree must be viewable and alterable.

- C6.2 $p-type(D) = DIRECT$. D must be a directory

$$P6.1 \quad \forall f_1 \quad CLS'(f) = \begin{cases} UNDEFINED & \text{if } C6.1 \wedge C6.2 \\ f_1 \in branch^*(map(F)) & (p \mu par(f_1)) \\ CLS(f_1) & \text{otherwise} \end{cases}$$

The classifications of all files in the subtree must be set to UNDEFINED. Note that the S_1 axioms guarantee that all files f_1 which are in the subtree are at classifications greater than or equal to $\underline{d}$ and thus may be altered.

$$P6.2 \quad \forall f_1 \quad ACL'(f_1) = \begin{cases} UNDEFINED & \text{if } C6.1 \wedge C6.2 \wedge (p \mu par(f_1)) \\ f_1 \in branch^*(map(F)) \\ ACL(f_1) & \text{otherwise} \end{cases}$$

$$P6.3 \quad \forall f_1 \quad TYPE'(f_1) = \begin{cases} UNDEFINED & \text{if } C6.1 \wedge C6.2 \wedge (p \mu par(f_1)) \\ f_1 \in branch^*(map(F)) \\ TYPE(f_1) & \text{otherwise} \end{cases}$$

$$P6.4 \quad \forall f_1 \quad RINGB'(f_1) = \begin{cases} UNDEFINED & \text{if } C6.1 \wedge C6.2 \wedge (p \mu par(f_1)) \\ f_1 \in branch^*(map(F)) \\ RINGB(f_1) & \text{otherwise} \end{cases}$$

$$P6.5 \quad \forall f_1 \quad D-char'(f_1) = \begin{cases} UNDEFINED & \text{if } C6.1 \wedge (p \mu f_1) \\ f_1 \in branch^*(map(F)) \\ D-char(f_1) \end{cases}$$

$$P6.6 \quad \forall f_1 \quad L-char'(f_1) = \begin{cases} UNDEFINED & \text{if } C6.1 \wedge (p \mu f_1) \\ [\exists N \text{ s.t. } f_1 = branch^*(map(F), N)] \\ L-char(f_1) & \text{otherwise} \end{cases}$$

$$spused'(L'char(map(D))) = 0 \quad (p \cup par(d))$$

All other attributes in the subtree are set to UNDEFINED, however, we have to be careful about QUOTA since it must move back up the tree. Actually, it is the SPUSED (spaced used) attribute that is affected and as shown in Section 6.6, SPUSED for file d can be set to zero without transferring any information.

P6.7-8 Unchanged

Clearly, all processes which are ATTACHED to any of the files in the subtree must be TERMINATED. One thing that must be demonstrated is that having the Security Kernel detach all processes attached to the tree does not result in a downward flow of information.

Clearly, any process p_1 , attached to some file F_1 in the subtree with root F must be at a clearance greater than or equal to the classification of D ; i.e., $CLS(D) \leq p-clr(p_1)$. This is a result of the fact that

$$map(F) \delta map(F_1)$$

that is F_1 is in the subtree.

Therefore:

$$\begin{aligned} map(D) &\equiv map(alpar(F)) \delta map(alpar(F_1)) \Rightarrow \\ CLS(D) &\leq CLS(alpar(F_1)) \end{aligned}$$

where D is F 's parent.

Also:

$$CLS(alpar(F_1)) \leq p-clr(p_1)$$

that is, p_1 must be able to view F_1 's parent. In order for p to

perform "DESTROY SUBTREE whose root is $\underline{F}$," it must be able to view and alter D , hence $p\text{-clr}(p) = \text{CLS}(d)$.

Combining these restrictions, we have:

$$p\text{-clr}(p) = \text{CLS}(D) \trianglelefteq \text{CLS}(\text{alpar}(F_1)) \trianglelefteq p\text{-clr}(p_1)$$

Thus changes to another process' PST will not result in a security compromise.

$$\text{P6.9} \quad \forall F_1 \in \text{PST}, p\text{-type}'(F_1) = \begin{cases} \text{UNDEFINED if } C6.1 \wedge C6.2 \wedge \\ \text{map}(F_1) \in \text{branch}^*(\text{map}(F)) & (p \mu \text{PST}) \\ p\text{-type}(F_1) \text{ otherwise} \end{cases}$$

$$\text{P6.10} \quad \forall F_1 \in \text{PST}, \text{alpar}'(F_1) = \begin{cases} \text{UNDEFINED if } C6.1 \wedge C6.2 \wedge \\ \text{map}(F_1) \in \text{branch}^*(\text{map}(F)) & (p \mu \text{PST}) \\ \text{alpar}(F_1) \text{ otherwise} \end{cases}$$

$$\text{P6.11} \quad \forall F_1 \in \text{PST}, \text{childcnt}'(F_1) = \begin{cases} \text{UNDEFINED if } C6.1 \wedge C6.2 \wedge \\ \wedge \text{map}(F_1) \in \text{branch}^*(\text{map}(F)) & (p \mu \text{PST}) \\ \text{childcnt}(F_1) \text{ otherwise} \end{cases}$$

$$P6.12 \quad \forall F_1 \in PST, \text{ attached}'(F_1) = \begin{cases} \text{UNDEFINED if } C6.1 \wedge C6.2 \wedge \\ \text{map}(F_1) \in \text{branch}^*(\text{map}(F)) \\ \quad (p \in PST) \\ \text{attached}(F_1) \text{ otherwise} \end{cases}$$

$$F_1 \in PST, \text{ view-attached}'(F_1) \text{ iff } \text{view-attached}(F_1) \\ \wedge \text{ attached}'(F_1)$$

$$F_1 \in PST, \text{ alter-attached}'(F_1) \text{ iff } \text{alter-attached}(F_1) \\ \wedge \text{ attached}'(F_1)$$

$$P6.13 \quad \forall F_1 \in PST, \text{ map}'(F_1) = \begin{cases} \text{UNDEFINED if } C6.1 \\ \text{map}'(F_1) \in \text{branch}^*(\text{map}(F)) \\ \quad (p \in PST) \\ \text{map}'(F_1) \text{ otherwise} \end{cases}$$

$$P6.14 \quad \forall F_1 \in PST, \text{ p-entyno}'(F_1) = \begin{cases} \text{UNDEFINED if } C6.1 \wedge \\ \text{map}'(F_1) \in \text{branch}^*(\text{map}(F)) \\ \quad (p \in PST) \\ \text{p-entyno}(F_1) \text{ otherwise} \end{cases}$$

All of the process' information about the deleted files are set to UNDEFINED. Actually, this isn't necessary as long as the attached attributes become false, however, it is an additional precaution for reliability.

$$P6.15 \quad \forall f_1 \in F, \forall n \in \mathbb{N}, \text{branch}'(f_1, n) =$$

$$\begin{cases} \text{UNDEFINED if } f_1 \in \text{branch}^*(\text{map}(F), N) \\ \quad \wedge C6.1 \wedge C6.2 & (p \neq \text{par}(f_1)) \\ \text{branch}(f_1, n) \text{ otherwise} \end{cases}$$

All files in the subtree have these attributes set to null.

7.) ADD to ACL for file F [user U_i with permission A_i (for $i = 1, n$)]

This operation deals with the discretionary controls for a file.

Using this operation, one or more users with view and/or alter permission can be added to the Access Control List of a file.

$$C7.1 \quad \text{alter-attached}(D) = \text{TRUE} \quad (p \neq \text{PST})$$

Since d contains file f 's ACL attribute, we must be able to alter d .

$$C7.2 \quad p\text{-type}(D) = \text{DIRECT} \quad (p \neq \text{PST})$$

Only directories can contain the ACL attribute.

$$P7.1 \quad \forall f_1 \in F, \text{CLS}'(f_1) = \text{CLS}(f_1) \quad (-)$$

The file's classification does not change.

$$\begin{aligned} P7.2 \quad \forall f_1 \in F, \text{mayview}'(\text{ACL}(U_i, f_1)) &= \text{mayview}(\text{ACL}(U_i, f_1)) \\ &\vee [A_i.\text{alter} \wedge C7.1 \wedge C7.2] \quad (p \neq d) \end{aligned}$$

If desired, view permission is given to user U_i for file f_1 .

$$\begin{aligned} \forall f_1 \in F, \text{mayalter}'(\text{ACL}(U_i, f_1)) &= \text{mayalter}(\text{ACL}(U_i, f_1)) \\ &\vee [A_i.\text{alter} \wedge C7.1 \wedge C7.2] \quad (p \neq d) \end{aligned}$$

Similarly, alter permission can be given.

P7.3-14 No change.

8.) REMOVE from ACL for file F [user U_i permission A_i for $i = 1, n$]

The REMOVE ACL operation is used to terminate access privileges for one or more users to the specified file.

C8.1 $view-attached(D) = TRUE$

$alter-attached(D) = TRUE$

The operation requires that a file be checked for a given user and the specified access and that it be changed.

P8.1 $\forall F_1 \in PST, CLS'(F_1) = CLS(F_1)$

P8.2 $\forall f_1 \in F \text{ mayview}'(ACL(U_i, f)) = \text{mayview}(ACL(U_i, f)) \quad (p \mu \text{par}(f))$

$n \neg [A_i.view \wedge C8.1 \wedge f_1 = \text{map}(F)]$

$\forall f_1 \in F \text{ mayalter}'(ACL(U_i, f)) = \text{mayalter}(ACL(U_i, f)) \quad (p \mu \text{par}(f))$

$n \neg [A_i.alter \wedge C8.1 \wedge f_1 = \text{map } F]$

Changes are made to a file's ACL as requested.

P8.3-8 Unchanged.

In a manner similar to that used for DESTROY SUBTREE, various users of file F must be forcefully detached in order to guarantee that a user can't get access due to an obsolete capability.

P8.9-11 Unchanged.

P8.12 $\forall p_1 \in P, \forall F_1 \in PST \text{ } p_1 \text{ view-attached}'(F_1) = \text{view-attached}_{p_1}(F_1)$

$n \neg [A_i.view \wedge C8.1 \wedge (p\text{-user}(p_1) \in U_i) \wedge F_1 = F]$

$(p \mu PST_1)$

$\forall p_1 \in p, \forall F_1 \in PST \text{ } p_1 \text{ alter-attached}'(F_1) = \text{alter-attached}_{p_1}(F_1)$

$n \neg [A_i.alter \wedge C8.1 \wedge (p\text{-user}(p_1) \in U_i) \wedge F_1 = F]$

$(p \mu PST)$

$$\forall p_1 \in P, \forall F_1 \in PST \quad p_1, \text{ attached}' p_1(F_1) = \text{view-attached}' p_1(F_1) \\ \cup \text{ alter-attached}' p_1(F_1)$$

All processes which are attached to file F and are affected by the operation are changed as specified above.

$$\begin{aligned} \text{P8.13} \quad \forall F_1 \in PST, \forall p_1 \in P, \text{ map}' p_1(F_1) & \quad (p \in PST) \\ & = \begin{cases} \text{map } p_1(F_1) & \text{if } \text{attached}' p_1(F_1) \\ \text{UNDEFINED} & \text{otherwise} \end{cases} \end{aligned}$$

This property uses the same conditions as the "attached" function to determine whether there is a mapping into the file system.

$$\text{P8.14} \quad \forall p_1 \in P, p\text{-entryno}'(p_1) = p\text{-entryno}(p_1)$$

$$\begin{aligned} \text{P8.15} \quad \forall F_1 \in PST, \forall N \in N \quad \text{branch}'(\text{map}(F_1), N) & \quad (p \in PST) \\ & = \begin{cases} \text{branch}(\text{map}(F_1), N) & \text{if} \\ \neg [\text{map}(F_1) \in \text{branch}^*(\text{map}(F))] & \\ \text{UNDEFINED} & \text{otherwise} \end{cases} \end{aligned}$$

9.) RAISE CLEARANCE to C

This operation allows a process to raise its clearance. It is important to note that this is the only operation which may do that. We have made (perhaps unnecessarily) the assumption that a user has a maximum clearance that cannot be exceeded by any of his processes.

$$C9.1 \quad p\text{-clr}(p) \leq C \leq \max_{p \in \text{PST}} (p\text{-user}(p)) \quad (p \in \text{PST})$$

A user may not exceed his highest clearance.

$$P9.1-6 \quad \text{No change to file system.} \quad (-)$$

$$P9.7 \quad \forall p_1 \in P \quad p\text{-user}'(p_1) = p\text{-user}(p) \quad (-)$$

$$P9.8 \quad \forall p \in P \quad p\text{-clr}'(p) = \begin{cases} C & \text{if } (p_1=p) \wedge C9.1 \\ p\text{-clr}(p_1) & \text{otherwise} \end{cases} \quad (p \in \text{PST})$$

The clearance is set as specified.

$$P9.12 \quad \forall F_1, \text{alter-attached}'(F_1) = \begin{cases} \text{alter-attached}(F_1) \wedge (C \leq p\text{-cls}(F_1)) & p \in \text{PST} \\ \text{true} & \text{otherwise} \end{cases} \quad \vee \neg C9.1$$

The new clearance must be less than or equal to the classification of any file which remains alter-attached.

$$P9.13,14 \quad \text{map, } p\text{-entryno} \text{ are unchanged.}$$

$$P9.15 \quad \text{branch unchanged.}$$

10.) RAISE CLASS of file F to C

This operation is used to raise the classification of a file. As will be pointed out, there are several restrictions on the possible classifications for files in a user's subtree.

C10.1 $view-attached(D) = TRUE$ $(p \in PST)$

$alter-attached(D) = TRUE$ $(p \in PST)$

$p-type(F) = DATASEG$ OR $view-attached(F)=TRUE$ $(p \in PST)$

The process must be able to view and alter the file's directory. Also, in the case where F is a directory, F must be viewable. However, further examination of this condition reveals that it is a serious constraint since file f and its parent, d, must be at the same classification. Briefly, this is because

- 1) the process p must be able to view and alter file d
and hence $CLR(p) = CLS(d)$
- 2) p must view file f and thus $CLS(f) \leq CLR(p)$
- 3) and finally, the tree axioms of S, force the clearances to increase going away from the root, giving $CLS(d) \leq CLS(f)$

Combining all this information, we have:

$CLS(f) \leq CLR(p) = CLS(d) \leq CLS(f)$ or $CLS(f) = CLS(d)$

C10.2 $\forall n \in \mathbb{N}, C \leq cls(branch(map(F), n))$ OR $p-type(F) = DATASEG$

P10.1 $\forall F_1 \in PST \ CLS'(F_1) = \begin{cases} C & \text{if } C10.1 \wedge C10.2 \wedge F_1 = F \\ CLS(F_1) & \text{otherwise} \end{cases}$

The classification of the file specified is changed if the conditions are met.

P10.2-11 Unchanged.

$$\begin{aligned}
\text{P10.12} \quad \forall p_1 \in P, \forall F_1 \in \text{PST} \quad \text{view-attached}'_{p_1}(F_1) = \\
\begin{cases} \text{view-attached}_{p_1}(F_1) & \text{if } \neg [C \leq p\text{-cls}_{p_1}(F)] \\ \text{FALSE} & \text{otherwise} \end{cases} \\
\forall p_1 \in P, \forall F_1 \in F \quad \text{alter-attached}'_{p_1}(F_1) = \\
\begin{cases} \text{alter-attached}_{p_1}(F_1) & \text{if } \text{view-attached}'(D) \\ \text{FALSE} & \text{otherwise} \end{cases}
\end{aligned}$$

In order to remain alter attached all ancestral files must be viewable.
 (It is sufficient to show that the parent is viewable).

$$\begin{aligned}
\text{P10.13} \quad \forall p_1 \in P, \forall F_1 \in F \quad \text{map}'_{p_1}(F_1) = \\
\begin{cases} \text{map}'_{p_1}(F_1) & \text{if } \text{attached}'_{p_1}(F_1) \\ \text{NULL} & \text{otherwise} \end{cases}
\end{aligned}$$

The "map" function has the same constraints as the "attached" function.

P10.14 *p-entryno* is unchanged

P10.15 *branch* is unchanged.

7 CONCLUSION

7.1 Introduction

We have demonstrated the use of a series of mathematical structures to yield the specifications of a software system to supply what we term security of information. The use of this technique yields both the definition of the global properties the system is to have and the local assertions which must be proven about the implementation of the system. This technique is used to specify a complete protection system so that mistakes in the design can be avoided. The advantage of this approach is that the assertions are obtained before the implementation and thus can be used as a guide. This approach is likely to result in a provable implementation since the proof development can proceed concurrently with the implementation.

7.2 Summary

Chapter 2 presented S_0 , the first in a series of mathematical structures, whose purpose is to give a systematic presentation of the specification for a secure computing system based on military requirements. This first structure defines security at an abstract level. Since it is devoid of actual system implementation considerations, it is simple enough to be accepted intuitively as a definition of security. A basic security theorem was then proven in order to show the design of the structure was sound.

Chapter 3 presented structure S_1 as a refinement of the original structure S_0 . Structure S_1 is one step closer to specifying an actual system. A tree structured file system and a mailbox mechanism for interprocess communication were introduced. The assumptions about S_1 were used to logically determine the relationship between classifications of objects and their positions in the file system tree. It was then shown that S_1 is an interpretation of S_0 by proving that all theorems true in S_0 were also true in S_1 .

Chapter 4 specified a dynamic structure S_2 to describe changes in the state of the system. With proper constraints S_2 was shown to be an example of an S_1 structure by use of an intermediate structure $S_{1.5}$. The requirements of primitive commands changing the state of a S_2 system were specified and proved.

Chapter 5 developed a further refinement of S_2 as S_3 to allow specification of attributes of objects in the structure. Attributes were logically located in S_2 objects for the purpose of security but may be actually implemented differently. A primitive set of

Multics-like events that would be supported by a system were given together with the assertions and proofs necessary for security. Chapter 6 defined and outlined proofs of commands to be provided by a secure system. These commands forming S_4 provide all the attributes and operations necessary for the construction of a Multics-like operating system.

7.3 Further Work to be Done

Our work thus far has shown it is feasible to completely specify a security system. S_3 is not complete: there are ways yet to be specified to affect and observe effects on the system. These include billing, input/output, file backup and recovery, and paging.

7.3.1 More Structures

The next difficult problem is not the addition of more structures but the implementation of what has been specified. This development process would then become structured programming and we would be concerned with the assertions necessary to prove the implementation correct with respect to the assertions given by the previous level or the last structure.

7.4 Other Problems

Construction and proof of a software security system is not the entire problem. Ultimately the implementation depends on the hardware which may have unforeseen affects under certain conditions. Accurate specifications of what actually happens must be used in proofs about the implementation. In particular, the security system depends on some ad hoc integrity mechanism supplied by hardware to protect itself. This mechanism must be checked to make sure there is no way in which it can be bypassed by those outside the security system or bypassed by incorrect use by the security system.

Second, the computer must be physically safe from external tampering so that the system cannot be altered and its communications lines are safe.

Another problem is that the security system depends on correctly determining the privileges of a user. Some reliable means of authenticating the users identity or determining his clearance is therefore necessary.

7.5 Future Topics

This project was constrained to yield a system as much like Multics as possible. The attempt was to secure Multics with as few changes as possible, although some changes not visible to the user were made in order to make the system smaller. An alternative would be to design a new system concentrating on minimizing its complexity and thereby making the proof of correctness of the implementation more manageable.

Another extension to the structures could allow the set of classifications to vary. Our current design which assumes a fixed set of classifications is suitable for individual users like the Department of Defense which have a permanent set of classifications. However, a computer utility needs something more flexible so that a customer can protect his information from other customers and also control security between his various projects.

As mentioned previously, implementation would use an ad hoc integrity mechanism to protect itself from external interference. The concept of integrity could be formalized and combined into the structures to yield the required properties of a consistent integrity mechanism.

Another use of structured specification could be to formalize the ideas of domains and capabilities and develop the required properties of the mechanisms rather than simply to propose various mechanisms as is now done. Domains, in particular, have the advantage of partitioning the implementation of a security and integrity system, simplifying its proof.

REFERENCES

- [1] Ames, S. R., Jr., "File Attributes and Their Relationship to Computer Security", ESD-TR-74-191, M.S. Thesis, Case Western Reserve University, June, 1974.
- [2] Anderson, James P., "Computer Security Technology Planning Study", Vols. I and II, ESD-TR-73-51, Electronic Systems Division/Air Force Systems Command, L. G. Hanscom Field, Bedford, Mass., October, 1972.
- [3] Biba, K. J. et. al. "A Preliminary Specification of a Multics Security Kernel", The MITRE Corporation, Bedford, Mass. 1975.
- [4] Bell, D. E. and LaPadula, L. A., "Secure Computer Systems", Vols. I, II, and III, The MITRE Corporation, Bedford, Mass., 1973.
- [5] Knuth, Donald E., "The Art of Computer Programming", Vol. 1, Addison Wesley, 1968.
- [6] Lampson, Butler W., "A Note on the Confinement Problem", Communications of the ACM 16, 10 (October 1973), 613-615.
- [7] Lampson, Butler W., "Protection", Proceedings Fifth Annual Princeton Conference on Information Sciences and Systems, Dept. of Elect. Eng., Princeton University, Princeton, New Jersey, March, 1971, pp. 437-443.
- [8] MacLane, S. and Birkhoff, G., Algebra, The MacMillan Company, London, 1967, p. 489.
- [9] Organic, E. I., The Multics System: An Examination of Its Structure, M.I.T. Press, 1972.
- [10] Popek, Gerald J., "Access Control Models", "ESD-TR-106, ESD/AFSC, L. G. Hanscom Field, Bedford, Mass., February, 1973.
- [11] Schroeder, Michael and Saltzer, Jerome, "A Hardware Architecture for Implementing Protection Rings", Communications of the ACM, Vol. 15, No. 3., March, 1972.
- [12] Schell, Roger R., Downey, D. J., and Popek, G. J., "Preliminary Notes on the Design of Secure Military Computer Systems", ESD-MCI-73-1, ESDIAFSC, L.G. Hanscom Field, Bedford, Mass., January, 1973.
- [13] Statement of Work, "Secure Operating System Design", (Private Communication), The Mitre Corporation, 1973.

REFERENCES (continued)

- [14] Walter, K. G., et al. "Modeling the Security Interface", C.W.R.U., Jennings Computing Center Report No. 1158, Cleveland, Ohio, August, 1974.
- [15] Walter, K. G., et al. "Primitive Models for Computer Security", ESD-TR-74-117, C.W.R.U., January, 1974.
- [16] Weissman, Clark, "Security Controls in the ADEPT-50 Time-Sharing System", Proc. AFIPS 1969, Fall Joint Computer Conference, pp. 119-133.

Appendix A: A General Specification for Information Passing Systems, S_{-1}

Last moment reflections upon the overall structure of the sequence of specifications developed in this report led to the discovery that there is still a more primitive underlying conception of the system than even the S_0 specification presents. Its existence is hinted at by the "information passing" theorem proved in Chapter 2.

A.1 A Formal Description of S_{-1}

In the information passing specification there is only one underlying set:

Ac is the set of Accessories to information passing.

There are two relations involving information passing between accessories.

$\alpha \subseteq Ac \times Ac$ is the "allowed to pass information" relation between Accessories. ($\underline{x} \alpha \underline{y}$ means that accessory $\underline{x}$ is allowed to pass information to accessory $\underline{y}$)

$\rho \subseteq Ac \times Ac$ is the "might pass information" relation between accessories. ($\underline{x} \rho \underline{y}$ means that accessory $\underline{x}$ has some possible way to pass information to accessory $\underline{y}$.)

There are three axioms in this specification, and the first two reflect obvious facts about possible information passing.

A-1.1 (reflexivity): For $\underline{x} \in Ac$, $\underline{x} \rho \underline{x}$. (Accessory $\underline{x}$ may remember information that it already knows).

A-1.2 (transitivity): For any $\underline{x}, \underline{y}, \underline{z} \in Ac$, if $\underline{x} \rho \underline{y}$ and $\underline{y} \rho \underline{z}$, then $\underline{x} \rho \underline{z}$. (If accessory $\underline{x}$ might pass information to accessory $\underline{y}$ and $\underline{y}$ in turn might pass information to accessory $\underline{z}$, then we must take into

account the possibility that $\underline{x}$ may be passing information to $\underline{z}$.)

The third axiom connects possibility with permission.

A-1.3 (conformity): For $\underline{x}, \underline{y} \in Ac$, $\underline{x} \rho \underline{y} \Rightarrow \underline{x} \alpha \underline{y}$ (It should only be possible for accessory $\underline{x}$ to pass information to accessory $\underline{y}$ if $\underline{x}$ is allowed to pass information to $\underline{y}$.)

This last axiom is perhaps most perspicuously stated in contrapositive form:

Corollary: If accessory $\underline{x}$ is not allowed to pass information to accessory $\underline{y}$, then there must be no way for $\underline{x}$ to pass information to $\underline{y}$.

The "might pass information" relation ρ in this specification emphasizes the importance in subsequent specifications of identifying all possible paths by which information might pass from one point to another. The "allowed to pass information" relation α will be specified precisely by the Clearance/Classification scheme introduced in S_0 .

Our next task is to show how the S_{-1} and S_0 specifications fit together.

A.2 Proving That S_0 is a Possible Interpretation of S_{-1}

First we must identify the set Ac and the relations α and ρ of the S_{-1} specification using the sets, functions, and relations of the S_0 specification. Recall that in S_0 we have: a set of agents A , a set of repositories R , a set of security classes C , a "can observe" relation θ , a "can modify" relation μ , an "of lower security class" relation $\leq$, a clearance function Clr , and a classification function Cls .

The information passing accessories are clearly the agents A and the repositories R . So we let $Ac_0 = A \cup R$.

We clearly need the clearance and classification functions to define the "allowed to pass information" relation α . The definition will be simpler if we define an auxillary "Classing" function $Cl: A \cup R \rightarrow C$ according to the following definition:

$$Cl(x) = \begin{cases} Clr(x) & \text{if } x \in A \\ Cls(x) & \text{if } x \in R \end{cases}$$

Now we can define the "allowed to pass information" relation α_0 on the accessories Ac_0 by $x \alpha_0 y \iff Cl(x) \leq Cl(y)$. The definition of the "might pass information" relation ρ must involve the "observe" and "modify" relations θ and μ in S_0 . In fact, the proper definition is most easily given if we first define an auxillary "information can directly transfer" relation τ on Ac_0 by

$$x \tau y \iff x \in A \text{ and } y \in R \text{ and } x \mu y \text{ or } \\ x \in R \text{ and } y \in A \text{ and } y \theta x$$

The "might pass information" relation ρ_0 is just the reflexive, transitive closure τ^* of the relation τ .

To show that, with these definitions, S_0 is a valid interpretation of the S_{-1} specification, it is necessary to verify that the three S_{-1} axioms hold. The first two axioms are trivial to verify, since ρ_0 is by definition the reflexive, transitive closure of a relation τ .

Theorem 1 (reflexivity): For $\underline{x} \in Ac_0$, $\underline{x} \rho_0 \underline{x}$.

Theorem 2 (transitivity): For $\underline{x}, \underline{y}, \underline{z} \in Ac_0$, if $\underline{x} \rho_0 \underline{y}$ and $\underline{y} \rho_0 \underline{z}$ then $\underline{x} \rho_0 \underline{z}$.

Verifying the third axiom, however, requires the use of the axioms of S_0 .

Theorem 3 (conformity): For $\underline{x}, \underline{y} \in Ac_0$, if $\underline{x} \rho_0 \underline{y}$, then $\underline{x} \alpha_0 \underline{y}$

Proof: The result is obtained by showing α_0 is a reflexive, transitive relation which includes the relation τ . Since τ^* is by definition the smallest reflexive, transitive relation which includes the relation τ , $\tau^* = \rho_0$ must be included in the relation α_0 , which is what this theorem asserts.

Lemma 1 (reflexivity): For $\underline{x} \in Ac_0$, $\underline{x} \alpha_0 \underline{x}$.

Proof: $Cl(\underline{x}) \trianglelefteq Cl(\underline{x})$, since $\trianglelefteq$ is reflexive
 $\therefore \underline{x} \alpha_0 \underline{x}$ by the definition of α_0 .

Lemma 2 (transitivity): For $\underline{x}, \underline{y}, \underline{z} \in Ac_0$, if $\underline{x} \alpha_0 \underline{y}$ and $\underline{y} \alpha_0 \underline{z}$, then $\underline{x} \alpha_0 \underline{z}$.

Proof: assume $\underline{x} \alpha_0 \underline{y}$ and $\underline{y} \alpha_0 \underline{z}$
 $Cl(\underline{x}) \trianglelefteq Cl(\underline{y})$ by definition of α_0
 $Cl(\underline{y}) \trianglelefteq Cl(\underline{z})$ by definition of α_0
 $Cl(\underline{x}) \trianglelefteq Cl(\underline{z})$ by the transitivity of $\trianglelefteq$
 $\therefore \underline{x} \alpha_0 \underline{z}$ by the definition of α_0

Lemma 3 (containment): $\tau \subseteq \alpha_0$. (That is, for $\underline{x}, \underline{y} \in Ac_0$, if $\underline{x} \tau \underline{y}$, then $\underline{x} \alpha_0 \underline{y}$)

Proof: assume $\underline{x} \tau \underline{y}$
either $\underline{x} \in A$ and $\underline{y} \in R$ and $\underline{x} \mu \underline{y}$
or $\underline{x} \in R$ and $\underline{y} \in A$ and $\underline{y} \theta \underline{x}$ by definition of τ .

Case 1: $\underline{x} \mu \underline{y}$

$Clr(\underline{x}) \trianglelefteq Cls(y)$ by the dissemination
axiom.

$Cl(\underline{x}) \trianglelefteq Cl(y)$ by definition of Cl

Case 2: $\underline{y} \theta \underline{x}$

$Cls(x) \trianglelefteq Clr(y)$ by the acquisition
axiom

$Cl(x) \trianglelefteq Cl(y)$ by definition of Cl

$Cl(\underline{x}) \trianglelefteq Cl(y)$ since this is true in either
case.

$\underline{x} \alpha_0 \underline{y}$ by definition of α_0 .

$x \tau y \Rightarrow x \alpha_0 y$ as desired

The S_0 specification has now been shown to be an example of the S_{-1} specification, and, by our general methodology, we know that all subsequent specifications will be examples of S_0 and hence of S_{-1} . Accordingly we know that all of our specifications will carry along this basic, very simple idea that information must not go where it is not allowed (by regulation) to go.

Appendix B: Formal Description of S_1 Consistency and the S_1 - S_0 Relationship

This appendix contains the propositions and proofs which are necessary to establish the relationship between the structures S_1 and S_0 .

PROPOSITION 3.1: S_1 is consistent.

PROOF: Consider the relational structure S_1' consisting of the set $\{a', b', c', d'\}$ with relations $\rho'_F, \sigma'_F, \rho'_M, \sigma'_M, \underline{\delta}', \delta'$ defined so that $d' \underline{\delta}' d', b' \delta' b', a' \sigma'_F b'$ hold, and these are the only relationships which hold, and functions Cls' and Clr' defined so that $Cls'(b') = Cls'(c') = Clr'(a') = d'$. Let i denote the one-to-one correspondence between the set $\{a', b', c', d'\}$ and any subset of four of the formal object symbols of S_1 . Further define the subsets A, F, M, C such that: $A = \{i(a')\}$, $F = \{i(b')\}$, $M = \{i(c')\}$ and $C = \{i(d')\}$. Also define the predicates in S_1' so that for $x, y \in \{a', b', c', d'\}$,

$$\begin{aligned} \text{if } x \rho'_F y & \quad \text{then} \quad i(x) \rho_F i(y) \\ \text{if } x \rho'_M y & \quad \text{then} \quad i(x) \rho_M i(y) \\ \text{if } x \sigma'_F y & \quad \text{then} \quad i(x) \sigma_F i(y) \\ \text{if } x \sigma'_M y & \quad \text{then} \quad i(x) \sigma_M i(y) \\ \text{if } x \underline{\delta}' y & \quad \text{then} \quad i(x) \underline{\delta} i(y) \\ \text{if } x \delta' y & \quad \text{then} \quad i(x) \delta i(y) \\ Cls(i(x)) & = i(Cls'(x)) \\ Clr(i(x)) & = i(Clr'(x)) \end{aligned}$$

It will now be shown that the axioms of S_1 hold true in S_1' .

A1.1 For all $c \in C$, $c = i(d')$, but $d' \underline{\delta}' d'$ in S_1' and thus

$$\underline{c \leq c}.$$

- A1.2 For all $c, d, e \in C$, $c = d = e = i(d')$ but $d' \triangleleft d'$ in S_1' and thus $c \triangleleft d$ and $d \triangleleft e$ implies $i(d') \triangleleft i(d')$ or $c \triangleleft e$.
- A1.3 For all $a \in A$, $f \in F$, $a = i(a')$ and $f = i(b')$, but $clr(a) = clr(i(a')) = i(clr'(a')) = i(d')$ and $cls(f) = cls(i(b')) = i(cls'(b')) = i(d)$ thus $a \rho_F f$ implies $cls(f) \triangleleft clr(a)$.
- A1.4 For all $a \in A$, $m \in M$ $a = i(a')$ and $m = i(c')$, but $clr(a) = i(d')$ and $cls(m) = i(d')$ as for A1.3, thus $a \rho_M m$ implies $cls(m) = clr(a)$.
- A1.5 Similar to A1.3.
- A1.6 Similar to A1.4.
- A1.7 For all $f \in F$, $f = i(b')$ and $b' \delta b'$ in S_1 thus $i(b') \delta i(b')$ and so $f \delta f$.
- A1.8 For all $f, g \in F$, $f = g = i(b')$ thus $f \delta g$ and $g \delta f$ implies $f = g$.
- A1.9 For all $f, g, h \in F$, $f = g = h = i(b')$ and since $b' \delta b'$ in S_1' , $i(b') \delta i(b')$, thus $f \delta g$ and $g \delta h$ implies $f \delta h$.
- A1.10 For all $f, g, h \in F$, $f = g = h = i(b')$, thus $g \delta f$ and $h \delta f$ implies $g \delta h$ or $h \delta f$.
- A1.11 For all $a \in A$ and $f, g \in F$, and $f = g = i(b')$ thus $a \rho_F g$ and $f \delta g$ implies $a \rho_F f$.
- A1.12 For all $a \in A$ and $f, g \in F$, $f = g = i(b')$ thus the hypothesis in the implication $a \sigma_F g$ and $f \delta g$ and $f = g$ implies $a \rho_F f$ can not hold true in S_1' . Therefore the implication holds no matter whether $a \rho_F f$ or not.

- A1.13 For all $a \in A$, $f, g \in F$, $f = g = i(b')$ thus $a \sigma_F f$ and $f \delta g$
implies $a \sigma_F g$.
- A1.14 For all $f \in F$, $f = i(b')$ and $a' \sigma' b'$ in S_1 thus there exists
 $a \in A$ namely $a = i(a')$ such that $a \sigma_F f$.

Because of the similarity between the axioms of S_0 and S_1 it would be simple to prove the counterpart of the basic theorem of S_0 directly in S_1 ; however, we prefer to prove the stronger result that every theorem of S_0 is a theorem of S_1 . To do this we will exhibit a mapping from the formal symbols of S_1 to the formal symbols of S_0 which preserves relationships. Informally we might view this as showing that S_1 is a valid interpretation of S_0 .

Proposition 3.2: There exists a one-to-one correspondence h from S_1 to S_0 which preserves relationships.

PROOF: Since the sets of formal symbols may be assumed to be equipotent, there will be a one-to-one correspondence h from the symbols of S_1 to the symbols of S_0 . We will show that the relational symbols in S_0 may be defined so that h preserves the relations. First, the subsets A , R , C , in S_0 are defined as follows:

$A_0 = h(A_1)$, $R = h(F \cup M)$, and $C_0 = h(C_1)$. (The use of subscripts 0 and 1 to indicate elements of S_0 and S_1 respectively should be clear. This convention will be maintained throughout the following.)

Second, the binary relational symbols θ , μ , $\leq$ in S_0 are defined so that for $a_0 \in A_0$, $r_0 \in R_0$, $c_0, d_0 \in C_0$,

(3.2.1) $a_0 \theta r_0$ if and only if $h^{-1}(a_0) \rho_F h^{-1}(r_0)$ or $h^{-1}(a_0) \rho_M h^{-1}(r_0)$,

(3.2.2) $a_0 \mu r_0$ if and only if $h^{-1}(a_0) \sigma_F h^{-1}(r_0)$ or $h^{-1}(a_0) \sigma_M h^{-1}(r_0)$,

(3.2.3) $c_0 \leq d_0$ if and only if $h^{-1}(c_0) \leq_1 h^{-1}(d_0)$.

(3.2.4) And finally, $cls_0(r_0) = cls_1(h^{-1}(r_0))$ and $cler_0(a_0) = cler_1(h^{-1}(a_0))$

(Note that since h is a one-to-one correspondence it has an inverse h^{-1} which is also a one-to-one correspondence.)

Proposition 3.3. Every valid interpretation of S_1 is also a valid interpretation of S_0 .

PROOF: Let the relational structure S be a valid interpretation of S_1 ; then there is a one-to-one correspondence i_1 from S to a subset of S_1 which preserves relations and such that the axioms of S_1 hold true in S under the interpretation. We claim that the composite function $i_0 = h_0 \circ i_1$ (where h is the function produced in Proposition 3.2 -- actually the restriction of h to the image of S under i_1 in S_1) from S to S_0 can be used to validly interpret S_0 in S . i_0 is a one-to-one correspondence since it is the composition of two one-to-one correspondences. Also the relationships in S are preserved under i_1 and h and thus under i_0 . It remains to be shown that the axioms of S_0 hold true in S .

It will be useful in the following to make the observation that if $a = i_0(m) = h(i_1(m))$ then since the one-to-one correspondence h has an inverse h^{-1} it is true that

$$(3.3.1) \quad h^{-1}(a) = i_1(m).$$

A0.1 For all $c \in C$ there exists an $m \in M$ such that $i_0(m) = c$.

Since axiom A1.1 holds true in S , $i_1(m) \leq_1 i_1(m)$; thus, since h preserves $\leq$, $h(i_1(m)) \leq_0 h(i_1(m))$. But this is equivalent to

saying $i_0(m) \leq_0 i_0(n)$ or $c \leq_0 c$.

A0.2 For all $c, d, e \in C_0$ there exist $m, n, p \in S$ such that
 $i_0(m) = c, i_0(n) = d$, and $i_0(p) = e$.
 $c \leq_0 d$ implies $i_1(m) \leq_1 i_1(n)$ in S_1 and $d \leq_0 c$ implies
 $i_1(n) \leq_1 i_1(p)$ in S_1 . Then since axiom A1.2 holds true in
 S , $i_1(m) \leq_1 i_1(p)$. Thus $h(i_1(m)) \leq_0 h(i_1(p))$ or $c \leq_0 e$.

A0.3 For all $a_0 \in A_0, r \in R$ there exist $m, n \in S$ such that
 $a_0 = i_0(m)$ and $r = i_0(n)$; thus, if $a_0 \theta r$ then two cases
are possible: (1) $i_1(m) \rho_F i_1(n)$ and (2) $i_1(m) \rho_F i_1(n)$.

Case 1. By axiom A1.3, $cls_1(i_1(n)) \leq_1 clr_1(i_1(m))$;
thus by 3.3.1, $cls_1(h^{-1}(r)) \leq_1 clr_1(h^{-1}(a_0))$,
and by 3.2.4 $cls_0(r) \leq_0 clr_0(a_0)$.

Case 2. By axiom A1.4, $cls_1(i_1(n)) = cls_1(i_1(m))$
which implies $cls_1(i_1(n)) \leq_1 clr_1(i_1(m))$
and the rest follows as in Case 1.

A0.4 For all $a_0 \in A_0, r \in R$ there exist $m, n \in S$ such that
 $a_0 = i_0(m)$ and $r = i_0(n)$; thus, if $a_0 \mu r$ then two cases
are possible (1) $i_1(m) \sigma_F i_1(n)$ and (2) $i_1(m) \sigma_M i_1(n)$.

Case 1. By axiom A1.5, $clr_1(i_1(m)) \leq_1 cls_1(i_1(n))$,
thus by 3.3.1, $clr_1(h^{-1}(a_0)) \leq_1 cls_1(h^{-1}(r))$,
and by 3.2.4, $clr_0(a_0) \leq_0 cls_0(r)$.

Case 2. By axiom A1.6 $clr_1(i_1(m)) \leq_1 cls_1(i_1(n))$
and so forth as in Case 1.

We note that one direct result of Proposition 3.3 is that the consistency of S_1 implies consistency of S_0 . Of more interest is the Corollary:

Corollary 2.4: Every theorem of S_0 holds true in every valid interpretation of S_1 .

PROOF: By Proposition 3.3 every valid interpretation of S_1 will also be a valid interpretation of S_0 in which every theorem holds true.

Specifically, the fundamental theorem T0.1 of S_0 holds true in every valid interpretation. Thus we can say that in any valid interpretation of S_1 if there is an information transfer path from a given mailbox or file to a second mailbox or file then the security class of the first mailbox or file is less than or equal to the security class of the second.

Appendix C: A Proof of Proposition 4.4.1

To show that every $S_{1.5}$ is an S_1 , we must identify all elements in S_1 in terms of the elements of $S_{1.5}$ and then demonstrate that the S_1 -axioms are satisfied by the $S_{1.5}$ structure under the identification. This will show that every $S_{1.5}$ structure is a logically valid interpretation of S_1 .

Given $S_{1.5} = \langle E_S, F_S, M_S, C, \underline{\Delta}, S_S, \gamma_S, \alpha_S, \beta_S, \omega_S, clr_S, cls_S, \pi \rangle$ where the components are defined as in lines (4.4.1) through (4.4.10) in terms of the underlying S_2 -structure, let $S' = \langle F', M', A', C', \rho_F', \sigma_F', \rho_M', \sigma_M', \leq', S, cls', clr' \rangle$ where the sets and relations are identified below.

$$\begin{array}{lll}
 F' = F_S, & M' = M_S, & A' = E_S, \\
 C' = C, & \rho_F' = \gamma_S, & \sigma_F' = \alpha_S, \\
 \rho_M' = \beta_S, & \sigma_M' = \omega_S, & \leq' = \underline{\Delta}_S, \\
 cls' = cls_S, & \text{and} & clr' = clr_S.
 \end{array}$$

We will then show that this S' satisfies the axioms of S_1 . In order to do this the following lemmas will be needed:

Lemma C.1: For all $\langle e, s \rangle, \langle e, t \rangle \in E_S$, $\langle e, s \rangle \pi^* \langle e, t \rangle$ implies $clr_S(\langle e, s \rangle) \leq clr_S(\langle e, t \rangle)$.

Proof: (By induction) Let $M = \{n \in \mathbb{N} \mid \langle e, s \rangle \pi^n \langle e, t \rangle \text{ implies}$

$\text{clr}_S(\langle e, s \rangle) \leq \text{clr}_S(\langle e, t \rangle) \text{ for all } \langle e, s \rangle, \langle e, t \rangle \in E_S\}$

First: $0 \in M$ since $\langle e, s \rangle \pi^0 \langle e, t \rangle$ implies $\langle e, s \rangle = \langle e, t \rangle$ and we have $\text{clr}_S(\langle e, s \rangle) = \text{clr}_S(\langle e, t \rangle)$.

Second: Suppose $n \in M$, then consider $n + 1$. If $\langle e, s \rangle$ and $\langle e, t \rangle \in E_S$ and $\langle e, s \rangle \pi^{n+1} \langle e, t \rangle$, then there exists an $\langle e, u \rangle \in E_S$ such that $\langle e, s \rangle \pi^n \langle e, u \rangle$ and $\langle e, u \rangle \pi \langle e, t \rangle$. By the inductive hypothesis the first implies $\text{clr}_S(\langle e, s \rangle) \leq \text{clr}_S(\langle e, u \rangle)$, and by definition the second implies $u \tau t$. In an S_1 -secure structure τ is permissible. Hence, by axiom A2.12, $\text{clr}_u(e) \leq \text{clr}_t(e)$; therefore by (4.4.5) $\text{clr}_S(\langle e, u \rangle) \leq \text{clr}_S(\langle e, t \rangle)$. The transitivity of $\leq$ (A2.16) gives the desired $\text{clr}_S(\langle e, s \rangle) \leq \text{clr}_S(\langle e, t \rangle)$. Thus, $n+1 \in M$. By induction $M = \mathbb{N}$, and the property holds for all $n \leq 0$. Hence the implication holds for $\pi^* = \bigcup_{n=0}^{\infty} \pi^n$.

The proofs of the next two lemmas are analagous and are omitted.

Lemma C.2 For all $\langle f, s \rangle, \langle f, t \rangle \in F_S$, $\langle f, s \rangle \pi^* \langle f, t \rangle$ implies $\text{cls}_S(\langle f, s \rangle) \leq \text{cls}_S(\langle f, t \rangle)$.

Lemma C.3 For all $\langle m, s \rangle, \langle m, t \rangle \in M_S$, $\langle m, s \rangle \pi^* \langle f, t \rangle$ implies $\text{cls}_S(\langle m, s \rangle) \leq \text{cls}_S(\langle m, t \rangle)$.

We now turn to the results needed to show $S_{1.5}$ is an example of S_1 . Axioms A1.1 and A1.2 are immediate consequences of axioms A2.15 and A2.16.

A1.3: For all $\langle e, s \rangle \in E_S$, $\langle f, t \rangle \in F_S$, $\langle e, s \rangle \vee_S \langle f, t \rangle$ implies $\text{cls}_S(\langle f, t \rangle) \leq \text{clr}_S(\langle e, s \rangle)$.

Proof: By (4.4.7), $\langle e, s \rangle \vee_S \langle f, t \rangle$ implies there exists $u \in S$ such that $e \in E_u$, $f \in F_u$, $e \vee_u f$, $\langle f, t \rangle \pi^* \langle f, u \rangle$ and $\langle e, u \rangle \pi^* \langle e, s \rangle$. By S_2 -security the state u is reachable from some secure state through permissible and (hence preserving) transitions; thus, u is statically secure, and $e \vee_u f$ implies $cls_u(f) \sqsubseteq clr_u(e)$ or equivalently $cls_S(\langle f, u \rangle) \sqsubseteq clr_S(\langle e, u \rangle)$. Also by lemma C.1 we have $clr_S(\langle e, u \rangle) \sqsubseteq clr_S(\langle e, t \rangle)$ and by lemma C.2 we have $cls_S(\langle f, t \rangle) \sqsubseteq cls_S(\langle f, u \rangle)$. Those results and axiom A7.16 (transitivity) gives $cls_S(\langle f, t \rangle) \sqsubseteq clr_S(\langle e, s \rangle)$.

The analogous proofs of the next three lemmas are mercifully omitted.

A1.4: For all $\langle e, s \rangle \in E_S$, $\langle m, t \rangle \in M_S$, $\langle e, s \rangle \beta_S \langle m, t \rangle$ implies $cls_S(\langle m, t \rangle) = clr_S(\langle e, s \rangle)$.

A1.5: For all $\langle e, s \rangle \in E_S$, $\langle f, t \rangle \in F_S$, $\langle e, s \rangle \alpha_S \langle f, t \rangle$ implies $clr_S(\langle e, s \rangle) \sqsubseteq cls_S(\langle f, t \rangle)$.

A1.6: For all $\langle e, s \rangle \in E_S$, $\langle m, t \rangle \in M_S$, $\langle e, s \rangle \omega_S \langle m, t \rangle$ implies $clr_S(\langle e, s \rangle) \sqsubseteq cls_S(\langle m, t \rangle)$.

A1.7: For all $\langle f, s \rangle \in F_S$, $\langle f, s \rangle \delta_S \langle f, s \rangle$

Proof: By (4.4.2) $f \in F_S$, and since every state in S_2 is statically secure, axiom A2.5 holds and implies $f \delta_S f$. Hence, by (4.4.4) we get $\langle f, s \rangle \delta_S \langle f, s \rangle$.

A1.8: For all $\langle f, s \rangle, \langle g, t \rangle \in F_S$, $\langle f, s \rangle \delta_S \langle g, t \rangle$ and $\langle g, t \rangle \delta_S \langle f, s \rangle$ implies $\langle f, s \rangle = \langle g, t \rangle$.

Proof: By (4.4.2) $f \in F_S$ and $g \in F_t$, and by (4.4.4) $s=t$, $f \delta_S g$, and $g \delta_S f$. But in the statically secure state,

axiom A2.6 implies that $f=g$; hence, $\langle f,s \rangle = \langle g,t \rangle$. ■

A1.9 For all $\langle f,s \rangle, \langle g,t \rangle, \langle h,u \rangle \in F_S$, $\langle f,s \rangle \delta_S \langle g,t \rangle$ and $\langle g,t \rangle \delta_S \langle h,u \rangle$ implies $\langle f,s \rangle \delta_S \langle h,u \rangle$.

Proof: Briefly, (4.4.4) implies $s=t=u$, $f \delta_S g$, and $g \delta_t h$, and axiom A2.7 implies $f \delta_S h$ which by (4.4.4) yields $\langle f,s \rangle \delta_S \langle h,u \rangle$. ■

A1.10 For all $\langle f,s \rangle, \langle g,t \rangle, \langle h,u \rangle \in F_S$, $\langle f,s \rangle \delta_S \langle g,t \rangle$ and $\langle h,u \rangle \delta_S \langle g,t \rangle$ implies $\langle f,s \rangle \delta_S \langle h,u \rangle$ or $\langle h,u \rangle \delta_S \langle f,s \rangle$.

Proof: By (4.4.4) $s=t=u$, $f \delta_S g$, and $h \delta_S g$, and in the statically secure state this means by axiom A2.8, $f \delta_S h$ or $h \delta_S f$. Thus, $\langle f,s \rangle \delta_S \langle h,u \rangle$ or $\langle h,u \rangle \delta_S \langle f,s \rangle$.

A1.11 For all $\langle e,s \rangle \in E_S$, $\langle f,t \rangle, \langle g,u \rangle \in F_S$ $\langle e,s \rangle \nu_S \langle g,u \rangle$ and $\langle f,t \rangle \delta_S \langle g,u \rangle$ implies $\langle e,s \rangle \nu_S \langle f,t \rangle$.

Proof: (4.4.4) and $\langle f,t \rangle \delta_S \langle g,u \rangle$ implies $t=u$ and $f \delta_t g$. $\langle e,s \rangle \nu_S \langle g,t \rangle$ implies by (4.4.7) there is a $v \in S$ such that $e \in E_v$, $g \in F_v$, $e \nu_v g$, $\langle e,v \rangle \pi^* \langle e,s \rangle$ and $\langle g,t \rangle \pi^* \langle g,v \rangle$. By A2.7 $g \in F_v$ and $f \delta_t g$ implies $f \in F_v$ and $f \delta_v g$. Then since state v is statically secure we have A2.9 $e \nu_v f$. Thus v is a state in which $e \in F_v$, $f \in F_v$, $e \nu_v f$ and $\langle e,v \rangle \pi^* \langle e,s \rangle$. If we can show that $\langle f,t \rangle \pi^* \langle f,v \rangle$, we will have the desired result $\langle e,s \rangle \nu_S \langle f,t \rangle$. By axiom A2.17, f exists in every state which contains g . If $\langle g,x \rangle \pi \langle g,y \rangle$ for states $x, y \in S$, it must be true that $f \in F_x$ and $x \tau y$; thus, $\langle f,x \rangle \pi \langle f,y \rangle$. By induction this extends to π^* ; therefore, $\langle g,t \rangle \pi \langle g,v \rangle$, and $f \delta_t g$ implies $\langle f,t \rangle \pi^* \langle f,v \rangle$. ■

The same technique gives:

A1.12 For all $\langle e, s \rangle \in E_S$, $\langle f, t \rangle, \langle g, u \rangle \in F_S$, $\langle e, s \rangle \alpha_S \langle g, u \rangle$,
 $\langle f, t \rangle \delta_S \langle g, u \rangle$, and $\langle f, t \rangle \neq \langle g, u \rangle$ implies $\langle e, s \rangle \vee_S \langle f, t \rangle$.

And finally we have:

A1.13 For all $\langle f, s \rangle, \langle g, t \rangle \in F_S$, $\langle f, s \rangle \delta_S \langle g, t \rangle$ implies
 $\text{cls}_S(\langle f, s \rangle) \leq \text{cls}_S(\langle g, t \rangle)$.

Proof: By (4.4.4) $\langle f, s \rangle \delta_S \langle g, t \rangle$ implies $s=t$ and $f \delta_S g$. Also,
 an S_2 -secure structure state s is statically secure; thus,
 $\text{cls}_S(f) \leq \text{cls}_S(g)$ which by (4.4.6) gives $\text{cls}_S(\langle f, s \rangle) \leq$
 $\text{cls}_S(\langle g, s \rangle)$.

Appendix D: Proofs Concerning the S_3 Events

We must show that the events presented in chapter 5 have sufficient conditions and properties to guarantee that security is not compromised. We will demonstrate that the axioms presented in sections 4.2-4 are satisfied. In other words, based on the assumption that the computation is in a secure state before the event, we will prove that the event occurs in accordance with the transition axioms (A2.12 - A2.17) and that the new state is locally secure (axioms A2.1 - A2.11).

Thus there are seventeen axioms to be proved for each of the seventeen events. Fortunately, however, it has been found that often, a given event has no effect on several of the axioms. For example the "Connect file" event in no way affects the mailboxes or the axioms concerned with them. In addition, a proof for some axiom from event x may be identical or extremely close to the arguments used for proving the same axiom from event y. We have therefore chosen to prove that each axiom holds for all events rather than prove that each event satisfies all axioms. In effect, there is a two dimensional array of proofs to be made (see figure D.1) and we have chosen to traverse rows first.

	ev1	ev2	ev3	...	ev17
axiom 2.1					
axiom 2.2					
⋮					
axiom 2.17					

Figure D.1

Axiom A2.1 For all $e_1 \in E, f_1 \in F, e_1 \vee f_1 \implies CLS(f_1) \trianglelefteq CLR(e_1)$.

We shall assume that the system is in some secure state. As a result, we can assume that axiom A2.1 is true before the event occurs.

We must now show that after the occurrence of any event, we have:

$$e_1 \vee' f_1 \implies CLS'(f_1) \trianglelefteq CLR'(e_1).$$

Case 1 E1 (e becomes view-connected to f). By P1.6 -

$e_1 \vee' f_1$ only if $e_1 \vee f_1$ OR $e_1 = \underline{e}$ and $f_1 = \underline{f}$ and the conditions are met.

i) $e_1 \vee f_1$.

$CLR'(e_1) = CLR(e_1)$ and $CLS'(f_1) = CLS(f_1)$ by P1.2 and P1.11. Using the assumption, $CLS'(f_1) = CLS(f_1) \trianglelefteq CLR(e_1) = CLR'(e_1)$.

ii) $e_1 = \underline{e} \wedge f_1 = \underline{f} \wedge C1.1 \wedge C1.2 \wedge C1.3$

$CLS'(f_1) \trianglelefteq CLR'(e_1)$ due to C1.2.

Case 2 E3 e is disconnected from f. By P3.6

If $e_1 \vee' f_1$ then $e_1 \vee f_1$. But $CLS'(f_1) = CLS(f_1)$ and $CLR'(e_1) = CLR(e_1)$ by P3.2 and P3.11. However, we obtain $CLS'(f_1) = CLS(f_1) \trianglelefteq CLR(e_1) = CLR'(e_1)$, using the assumption.

Case 3 E5 (destroy file f). By P5.6

We have $e_1 \vee' f_1$ only if $e_1 \vee f_1$ and $\neg(f \delta f_1 \wedge C5.1 \wedge C5.2)$. But this information means that $CLS'(f_1) = CLS(f_1)$ and $CLR'(e_1) = CLR(e_1)$ by P5.2 and P5.11 respectively. Using the assumption as in the previous case, $CLS'(f_1) \trianglelefteq CLR'(e_1)$.

Case 4 E7. (raise classification of f to C) By P7.6

We have $e_1 \vee' f_1$ only if $e_1 \vee f_1$ and $[\neg(f_1 = \underline{f} \wedge C7.1 \wedge C7.2 \wedge C7.3) \vee C \trianglelefteq CLR(e_1)]$ due to P7.6.

i) $\neg(f_1 = \underline{f} \wedge C7.1 \wedge C7.2 \wedge C7.3)$.

In this case $CLS'(f_1) = CLS(f_1)$ and arguments for Case 2 apply.

ii) $C \leq CLR(e_1)$, conditions are satisfied.

$CLS'(f_1) = C$ from P7.2 . Thus $CLS'(f_1) =$

$C \leq CLR(e_1) = CLR'(e_1)$.

Case 5

All other events.

Px.2 and Px.11 guarantee $CLS'(f_1) = CLS(f_1)$ and

$CLR'(e_1) = CLR(e_1)$. See Case 1 i for proof.

Axiom A2.2

For all $e_1 \in E, m_1 \in M, e_1 \beta m_1 \implies CLR(e_1) = M-CLS(m_1)$.

We must show $e_1 \beta' m_1 \implies CLR'(e_1) = M-CLS'(m_1)$.

Case 1

E9 (e becomes receive-connected to f). By P9.9:

$e_1 \beta' m$ only if $e_1 \beta m_1$ OR $(e_1 = \underline{e} \wedge m_1 = m \wedge C9.1)$.

i) $e_1 \beta m_1$.

But from the assumption $CLR(e_1) = M-CLS(m_1)$.

Using P9.7 and P9.11 we find that $M-CLS$ and

CLR are unchanged. Hence: $M-CLS'(m_1) = M-CLS(m_1) =$

$CLR(e_1) = CLR'(e_1)$.

ii) $e_1 = \underline{e} \wedge m_1 = m \wedge C9.1$.

From C9.1 we know $CLR(e_1) = M-CLS(m)$. Using the

arguments in part i), $M-CLS'(m_1) = CLR(e_1)$.

Case 2

E11 (e becomes disconnected from mailbox m).

From $e_1 \beta' m_1$ we know $e_1 \beta m_1$ by using P11.9. In a

manner similar to the previous case we show $M-CLS'(m_1)$

$= M-CLS(m_1) = CLR(e_1) = CLR'(e_1)$.

- Case 3 E12 (raise classification of $\underline{m}$ to $\underline{C}$) By P12.9:
 $e_1 \beta' m_1$ only if $\neg (m_1 = \underline{m} \wedge C12.1 \wedge C12.2)$
 From P12.7 we find $M-CLS'(m_1) = M-CLS(m_1)$. The argument proceeds as in case 1.
- Case 4 E16 (e raises its own clearance to C).
 $e_1 \beta' m_1$ only if $\neg (e_1 = \underline{e} \wedge C16.1)$ due to P16.9.
 But this means $CLR'(e_1) = CLR(e_1)$ due to P16.11. As in previous cases $M-CLS'(m_1) = CLR(e_1)$.
- Case 5 All other events.
 By Px.7 and Px.11 $M-CLS'(m_1) = M-CLS(m_1)$ and $CLR'(e_1) = CLR(e_1)$. See case 1 i.

Axiom A2.3 For all $e_1 \in E, f_1 \in F, e_1 \alpha f_1 \implies CLR(e_1) \trianglelefteq CLS(f_1)$.

We must show $e_1 \alpha' f_1 \implies CLR'(e_1) \trianglelefteq CLS'(f_1)$.

- Case 1 E2 (e becomes alter-connected to f). By P2.5:
 $e_1 \alpha' f_1$ only if $e_1 \alpha f_1$ OR $(e_1 = \underline{e} \wedge f_1 = f \wedge C2.1 \wedge C2.2 \wedge C2.3)$.
- i) $e_1 \alpha f_1$
 From P2.2 and P2.11 we get $CLS'(f_1) = CLS(f_1)$
 and $CLR'(e_1) = CLR(e_1)$. Using the assumption we obtain: $CLR'(e_1) = CLR(e_1) \trianglelefteq CLS(f_1) = CLS'(f_1)$.
- ii) $e_1 = \underline{e} \wedge f_1 = f \wedge C2.1 \wedge C2.2 \wedge C2.3$.
 From C2.2 we get $CLR(e_1) \trianglelefteq CLS(f_1)$. The rest of the argument follows i).
- Case 2 E3 (e becomes disconnected from $\underline{f}$).
 If $e_1 \alpha' f_1$ then $e_1 \alpha f_1$ due to P3.5. But as in previous cases, classifications and clearances don't change. Therefore $CLR'(e_1) \trianglelefteq CLS'(f_1)$.

Case 3

E5 (destroy file $\underline{f}$).

If $e_1 \alpha' f_1$ then $\neg(\underline{f} \delta f_1 \wedge C5.1 \wedge C5.2)$ by P5.5.

But this means $CLR'(e_1) = CLR(e_1)$ and $CLS'(f_1) = CLS(f_1)$. By the same reasoning as previous cases, $CLR'(e_1) \trianglelefteq CLS'(f_1)$.

Case 4

E16 ($\underline{e}$ raises its own clearance to $\underline{C}$)

From P16.5 we know $e_1 \alpha' f_1$ only if $e_1 \alpha f_1$ and $[C \trianglelefteq CLS(f_1) \text{ OR } \neg(C16.1 \wedge e_1 = e)]$

i) $\neg C16.1 \text{ OR } \neg e_1 = \underline{e}$

In this instance the clearance doesn't change and previous arguments apply.

ii) $C \trianglelefteq CLS(f_1)$

By P16.11 $CLR'(e_1) = C$ for $e_1 = \underline{e}$. By transitivity, $CLR'(e_1) = C \trianglelefteq CLS(f_1)$, but P16.2 guarantees that $CLS'(f_1) = CLS(f_1)$

Case 5

All other events.

From Px.2 and Px.11 we know $CLS'(f_1) = CLS(f_1)$ and $CLR'(e_1) = CLR(e_1)$. Arguments follow Case 1 i.

Axiom A2.4

For all $e_1 \in E, m_1 \in M, e_1 \omega m_1 \implies CLR(e_1) \trianglelefteq M-CLS(m_1)$.

We need to show that $e_1 \omega' m_1 \implies CLR'(e_1) \trianglelefteq M-CLS'(m_1)$, however, since the proofs are similar to those in axiom A2.3, they are omitted.

Axiom A2.5 For all f_1 in F , $f_1 \delta f_1$.

We must show that:

For all f_1 in F' , $f_1 \delta' f_1$

Case 1 All events except E4 (create file $\underline{f}$ in $\underline{d}$) and E5
(destroy file $\underline{f}$ in $\underline{d}$)

In each of these events we have, due to property X.4, $f_1 \delta f_2$
implies $f_1 \delta' f_2$. Substituting f_1 for f_2 gives: $f_1 \delta' f_1$.

Case 2 E4 (create file $\underline{f}$ in $\underline{d}$)

Clearly by P4.4, if $f_1 \delta f$ then $f_1 \delta' f_1$. The other
possibility is that $f_1 = f$ - the newly created file.
But also $f_1 = f_2 = \underline{f}$ implies $f_1 \delta' f_2$ which in turn
implies by substitution $f_1 \delta' f_1$. Hence for all
 $f_1 \in F'$, $f_1 \delta' f_1$.

Case 3 E5 (destroy file $\underline{f}$ in $\underline{d}$)

By definition, only those files which exist after the
event are members of F' . Thus " $ST'(f_1) = \text{USED}$ " is
equivalent to $f_1 \in F'$. Using this, property P5.1
(contrapositive) yields $\neg f \delta f_1$ (assuming that the
conditions are satisfied).

If some condition is not satisfied then by P5.4
 $f_1 \delta' f_2$ if and only if $f_1 \delta f_2$. By substitution,
 $f_1 \delta' f_1$ iff $f_1 \delta f_1$.

Continuing, from P5.4 we also get: if $f_1 \delta f$ and $\neg f \delta f_1$ then $f_1 \delta' f_1$.

Thus we have for all $f_1 \in F'$, if $f_1 \delta f_1$ then $f_1 \delta' f_1$.

Axiom 2.6 For all $f_1, f_2 \in F$, $f_1 \delta f_2 \wedge f_2 \delta f_1 \Rightarrow f_1 = f_2$.

Show that: for all $f_1, f_2 \in F'$ $f_1 \delta' f_2 \wedge f_2 \delta' f_1 \Rightarrow f_1 = f_2$.

Case 1 E4 (create file $\underline{f}$ in $\underline{d}$)

By property P4.4 if $f_1 \delta' f_2$ then either $f_1 \delta f_2$ OR $(f_1 \delta d \text{ and } f_2 = f)$ OR $f_1 = f_2 = \underline{f}$. Similarly $f_2 \delta' f_1$ implies $f_2 \delta f_1$ OR $(f_2 \delta \underline{d} \text{ and } f_1 = \underline{f})$ OR $f_2 = f_1 = f$.

i) Clearly if $f_1 \delta f_2$ **and** $f_2 \delta f_1$ then $f_1 = f_2$ by the assumption. Furthermore, by definition, $ST(f_1) =$

$ST(f_2) = \text{USED}$. Since by C4.4 $ST(f) = \text{UNUSED}$ we know that $f_1 \neq f \wedge f_2 \neq f$ for this subcase.

ii) $f_1 = f_2 = f$ satisfied trivially

iii) Finally we can show that the last subcase

$f_1 \delta d \wedge d \delta' f_2$ and $f_2 = f$ implies $\neg(f_2 \delta' f_1)$

For a proof by contradiction, suppose $f_2 \delta' f_1$.

Then since $d \delta f_2$ we find by transitivity $d \delta' f_1$.

Now $f_1 \delta d$ implies $f_1 \delta' d$ by P4.4. Using this

fact with $d \delta' f_2$ we get $f_1 \delta' f_2$. However by

C4.3 we have $\neg(d \delta' f_1 \wedge f_1 \delta' f)$ or substituting

$\neg(d \delta' f_1 \wedge f_1 \delta' f_2)$ - a contradiction.

Thus if $f_1 \delta' f_2$ and $f_2 \delta' f_1$ then either $f_1 = f_2 = \underline{f}$

OR $f_1 \delta f_2$ and $f_2 \delta f_1$ and $f_1 = f_2$.

Case 2

All other events.

If $f_1 \delta' f_2$ then $f_1 \delta f_2$ because of property PX.4.

Similarly $f_2 \delta' f_1$ implies that $f_2 \delta f_1$. But $f_2 \delta f_1$ and $f_1 \delta f_2$ yield the result $f_1 = f_2$ by the assumption.

A2.7

$f_1, f_2, f_3 \in F, f_1 \delta f_2 \wedge f_2 \delta f_3 \implies f_1 \delta f_3$

Show that

$f_1, f_2, f_3 \in F', f_1 \delta' f_2, f_2 \delta' f_3 \implies f_1 \delta' f_3.$

Case 1

E4 (create file $\underline{f}$ in d)

i) $f_1, f_2, f_3 \in F$ - By the assumption, $f_1 \delta f_2$,
 $\wedge f_2 \delta f_3 \implies f_1 \delta f_3$. Thus $f_1 \delta' f_3$ by P4.4.

ii) $f_1, f_2 \in F, f_3 = f, f \in F', \neg(f \in F)$. By P4.4,
 $f_2 \delta' f_3$ only if $f_2 \delta d$. By transitivity $f_1 \delta d$.
But $f_1 \delta d$ means that $f_1 \delta' f_3$ also due to P4.4
(assuming all conditions are met).

Lemma 1

if $f_1 = f \wedge f_1 \delta f_2$ then $f_1 = f_2 = f$. By P4.4b

$f \delta f_2$ implies $f = f_2$.

iii) $f_1 \in F, f_2 = f$.

Clearly if $f_2 = f$ and $f_2 \delta f_3$, then by lemma 1
we must have $f_2 = f_3 = \underline{f}$. Thus $f_1 \delta' f_3$ since
 $f_1 \delta' f_2$.

iv) $f_1 = \underline{f}$

By lemma 1, $f_1 = f_2 = f_3 = \underline{f}$. By axiom A2.5

$f_1 \delta' f_3$.

Case 2

all other events.

From $f_1 \delta' f_2$ and $f_2 \delta' f_3$ we obtain $f_1 \delta f_2$

and $f_2 \delta f_3$ from properties Px.4. Furthermore, the

transitivity assumption assures that $f_1 \delta f_3$. But using Px.4 again we get $f_1 \delta' f_3$.

Axiom A2.8 For $f_1, f_2, f_3 \in F$, $f_1 \delta f_3$ and $f_2 \delta f_3 \Rightarrow f_2 \delta f_1$
OR $f_1 \delta f_2$.

We must show $f_1 \delta' f_3 \wedge f_2 \delta' f_3 \Rightarrow f_1 \delta' f_2$ OR $f_2 \delta' f_1$.

Case 1 E4 (create file)

- i) $f_1, f_2, f_3 \in F$ is true by the assumption.
- ii) Given, $f_1, f_2 \in F$, $f_3 = \underline{f}$ and all conditions are met then $f_1 \delta' f_3$ implies $f_1 \delta d$ by P4.4. Similarly, if $f_2 \delta' f_3$ then $f_2 \delta d$. But combining these, we get $f_1 \delta f_2$ or $f_2 \delta f_1$ by the assumption.
From P4.4 we know $f_1 \delta' f_2$ or $f_2 \delta' f_1$
- iii) $f_2 = f$
Using Lemma 1, we know also that $f_3 = f$. Since $f_1 \delta' f_3$, we must also have $f_1 \delta' f_2$.
Similar arguments apply if $f_1 = \underline{f}$ or both f_1 and f_2 are equal to f .

Case 2 E5 destroy file $\underline{f}$ in d . From $f_1 \delta' f_3$ and $f_2 \delta' f_3$ we know $f_1 \delta f_3$, $f_2 \delta f_3$, $\neg(C5.1 \wedge C5.2 \wedge f \delta f_1)$ and $\neg(C5.1 \wedge C5.2 \wedge f \delta f_2)$ due to P5.4. From the assumption we know either $f_1 \delta f_2$ OR $f_2 \delta f_1$. Knowing this along with $\neg(C5.1 \wedge C5.2 \wedge f \delta f_1)$ and $(C5.1 \wedge C5.2 \wedge f \delta f_2)$ gives us $f_1 \delta' f_2$ OR $f_2 \delta' f_1$ using P5.4.

Case 3 All other events.

From $f_1 \delta' f_3$ and $f_2 \delta' f_3$ we know $f_1 \delta f_3$ and $f_2 \delta f_3$ by Px.4. From the assumption we know either $f_1 \delta f_2$

OR $f_2 \delta f_1$. Now using Px.4 we obtain $f_1 \delta' f_2$ OR $f_2 \delta' f_1$.

Axiom A2.9 For $e_1 \in E, f_1, f_2 \in F, e_1 \vee f_2, f_1 \delta f_2$ implies $e_1 \vee f_1$.

We must show $e_1 \vee' f_2, f_1 \delta' f_2$ implies $e_1 \vee' f_1$.

Case 1 $E_1 \underline{e}$ is view-connected to $\underline{f}$.

From P1.6 we get two possibilities.

i) $e_1 \vee' f_2, \neg(e_1 \vee f_2)$

From C1.3 we know $e \vee d$, e can-view $\underline{f}$'s parent.

And by the assumption, $f_1 \delta d$ implies $e \vee f_1$.

We must show that if $f_1 \delta' f$ then $f_1 \delta d$. When f was created, condition C4.3 guaranteed

$\neg(d \delta f_1 \wedge f_1 \delta' f)$ for any f_1 . Hence $f_1 \delta' f \wedge d \delta' f$ means that $f_1 \delta d$ because of C4.3 and

Axiom A2.8. Thus for all $f_2 = f$ and $e_1 = e$,

$f_1 \delta f_2$ and $e \vee f_2$ imply $e_1 \vee f_1$ and hence $e_1 \vee' f_1$.

ii) $e_1 \vee f_2, \neg(C1.1 \wedge C1.2 \wedge C1.3 \wedge e_1 = e \wedge f_1 = f)$

For all $e_1 \in E, f_1 \in F, e_1 \vee' f_2$ only if $e_1 \vee f_2$ due to P1.6. But if $e_1 \vee f_2$ and $f_1 \delta f_2$ then $e_1 \vee f_1$ by the assumption and hence $e_1 \vee' f_1$

Case 2 E3 (disconnect file f)

From P3.6, we know that $e_1 \vee' f_2$ only if $e_1 \vee f_2$ and

$\neg(e_1 = e \wedge f \delta f_2 \wedge C3.1)$. But since $f_1 \delta f_2$,

if $\neg(f \delta f_2)$ then $\neg(f \delta f_1)$. Hence $\neg(e_1 = e \wedge f \delta f_1 \wedge C3.1)$ must also be true. This along with

the assumption that $e_1 \vee f_1$ means $e_1 \vee' f_1$ by P3.6.

Case 3.

E5 (destroy file f)

Using P5.6, $e_1 \vee' f_2$ assures $e_1 \vee f_2$ and $\neg(f \delta f_2 \wedge C5.1 \wedge C5.2)$. Using reasoning similar to that in Case 2, we know $\neg(f \delta f_1 \wedge C5.1 \wedge C5.2)$ is also true. Also, $e_1 \vee f_2$ only if $e_1 \vee f_1$ because of the assumption. But now, $e_1 \vee' f_1$ also due to P5.6.

Case 4.

E7 (raise classification of f)

i) $\neg(f \delta f_2)$ (neither file is in f 's subtree)

$\neg(f \delta f_2)$ only if $f \neq f_2$. But this and $e_1 \vee f_2$ means $e_1 \vee' f_2$ due to P7.6. Since $f_1 \delta f_2$, we know $e_1 \vee f_1$ from the assumption. Furthermore, we can show $e_1 \vee' f_1$ by similar reasoning. Since $f_1 \delta' f_2$ only if $f_1 \delta f_2$, we have shown that $f_1 \delta' f_2$ and $e_1 \vee' f_2$ implies $e_1 \vee' f_1$ where f_1 and f_2 are not in f 's subtree.

ii) $f \delta f_2$ (at least one of the files is in f 's subtree.)

From $e_1 \vee' f_1$, P7.6 guarantees $\neg(f = f_1 \wedge C7.1 \wedge C7.2 \wedge C7.3)$ OR $C \leq CLR(e_1)$

We know from P7.2 that $CLS'(f_2) =$

$CLS(f_2)$. But by the data acquisition axiom (A2.1)

$CLS'(f_2) \leq CLR(e_1)$. And by axiom A2.11 $CLS(f_1)$

$\leq CLS(f_2)$. Furthermore, C7.3 guarantees that

$CLS'(f_1) \leq CLS(f_2)$ if f_1 happens to be f .

Therefore for any f_1 such that $f_1 \delta f_2$, $CLS'(f_1)$

$\leq CLS(f_2) \leq CLR(e_1)$. But this is sufficient (using

property P7.6) to show $e_1 \vee' f_1$.

Case 5

All other events.

Properties Px.2 and Px.11 guarantee $CLS'(f_1) =$

$CLS(f_1)$, $CLS'(f_2) = CLS(f_2)$ and $CLR'(e_1) = CLR(e_1)$.

See case 1 ii for proofs.

Axiom A2.10

For all $e_1 \in E$, $f_1, f_2 \in F$, $e \alpha f_2$, $f_1 \delta f_2$ and $f_1 \neq f_2$
 $\implies e_1 \vee f_1$.

We must show that $e_1 \alpha' f_2$, $f_1 \delta' f_2$ and $f_1 \neq f_2 \implies e_1 \vee' f_1$.

Case 1 E2

(e is alter connected to f)

From property P2.5, we know that $e_1 \alpha f_2$ OR ($e_1 = \underline{e} \wedge$
 $f_2 = \underline{f} \wedge C2.1 \wedge C2.2 \wedge C2.3$) if $e_1 \alpha' f_2$.

i) $e_1 \alpha f_2$

By the assumption, for all $f_1 \delta f_2$, $e_1 \vee f_1$.

But we know $e_1 \vee' f_1$ also.

ii) $e_1 = \underline{e} \wedge f_2 = \underline{f} \wedge C2.1 \wedge C2.2 \wedge C2.3$. From C2.3
we know $e \vee d$, (We defined d : $d \delta f_2$ and
 $\forall f_1 \delta' f_2$, $f_1 \delta d$, that is all files which do-
minate f_2 also dominate d .) But using axiom A2.9
we prove $e_1 \vee f_1$ given $e_1 \vee d$. But if $e_1 \vee f_1$
then $e_1 \vee' f_1$.

Case 2

E3 (disconnect e from file f).

From P3.5, $e_1 \alpha' f_2$ implies $e_1 \alpha f_2$ and $\neg(e_1 = e \wedge$
 $f \delta f_2 \wedge C3.1)$. Now $e_1 \alpha f_2$ and $f_1 \delta f_2$ and $f_1 \neq f_2$
means $e_1 \vee f_1$ because of the assumption. But since
 $\neg(e_1 = e \wedge f \delta f_2 \wedge C3.1)$ and $f_1 \delta f_2$ only if
 $\neg(e_1 = e \wedge f \delta f_1 \wedge C3.1)$, we know $e_1 \vee' f_1$ due
to property P3.6.

Case 3

E5 (destroy file $\underline{f}$) By P5.5

$e_1 \alpha' f_2$ implies $e_1 \alpha f_2$ and $\neg(f \delta f_1 \wedge C5.1 \wedge C5.2)$.

Using arguments similar to those in case 2, we find

$e_1 \vee f_1$ and furthermore $e_1 \vee' f_1$ using P5.5 and P5.6.

Case 4

E7 (raise classification of $\underline{f}$).

From P7.5 we know that $e_1 \alpha f_2$ and $[\neg(f \delta f_2 \wedge$

$f = f_2 \wedge C7.2 \wedge C7.3) \text{ OR } C \leq CLR(e_1)]$. The arguments

closely follow those for axiom A2.9 and shall only be

summarized: If neither file is in $\underline{f}$'s subtree, the

view relation remains the same. In fact as long as

f_1 is not in the subtree " e_1 can-view f_1 " is unchanged.

If f_1 is in the subtree (but it is not f) the change

in classification can not exceed f_1 's classification

so e_1 still can-view f_1 . If $f_1 = f$, then P7.5 guaran-

tees that f_1 remains at a compatible classification.

Case 5

E16 (e raises its own clearance)

$e_1 \alpha' f_2$ implies $e_1 \alpha f_2$ and hence for $f_1 \delta f_2$, $e_1 \vee f_1$.

But the "can-view" relation is unchanged hence $e_1 \vee' f_1$.

Case 6

All other events.

Properties Px.2 and Px.11 guarantee $CLS'(f_1) = CLS(f_1)$,

$CLS'(f_2) = CLS(f_2)$ and $CLR'(e_1) = CLR(e_1)$. See case

1 i for proof.

Axiom A2.11 $f_1, f_2 \in F, f_1 \delta f_2 \implies CLS(f_1) \sqsubseteq CLS(f_2).$

We must show $f_1 \delta' f_2 \implies CLS'(f_1) \sqsubseteq CLS'(f_2).$

Case 1 E4 (create file $\underline{f}$ in $\underline{d}$).

From P4.4, $f_1 \delta f_2$ OR $(f_1 \delta d \wedge f_2 = f \wedge C4.1$
 $\wedge \dots \wedge C4.6)$ OR $(f_1 = f_2 = \underline{f} \wedge C4.1 \wedge \dots C4.6).$

i) $f_1 \delta f_2$

$CLS'(f_1) = CLS(f_1) \sqsubseteq CLS(f_2) = CLS'(f_2).$

ii) $f_1 \delta d \wedge f_2 = f \wedge C4.1 \wedge \dots \wedge C4.6.$ From C4.3

$\neg(f_1 \delta' f_2 \wedge d \delta f_1).$ But this means $f_1 \delta d$
 due to axiom A2.4. We know $CLS(f_1) \sqsubseteq CLS(d)$ and
 from P4.2, $CLS'(d) = CLS(d)$ and $CLS'(f_1) = CLS(f_1).$

Furthermore C4.5 and P4.2 guarantee that $CLS(d)$
 $\sqsubseteq CLS'(f_2).$ Hence $CLS'(f_1) \sqsubseteq CLS'(f_2).$

iii) $f_1 = f_2 = \underline{f}$

$CLS'(f_1) = CLS'(f_2)$ and hence $CLS'(f_1) \sqsubseteq CLS'(f_2)$
 trivially.

Case 2 E5 (destroy file $\underline{f}$).

For all files f_2 in use ($ST(f_2) \neq \text{UNUSED}$), $\neg(f \delta f_2$
 $\wedge C5.1 \wedge C5.2).$ But also, $\neg(f \delta f_1 \wedge C5.1 \wedge C5.2).$
 Hence $CLS'(f_1) = CLS(f_1)$ and $CLS'(f_2) = CLS(f_2)$ by P5.2.
 Thus $CLS'(f_1) = CLS(f_1) \sqsubseteq CLS(f_2) = CLS'(f_2).$

Case 3 E7 (raise classification of $\underline{f}$).

i) $f_1 = \underline{f}$ and conditions are met.

$CLS'(f) = C$ due to P7.2. But for all f_2 such that
 $f_1 \delta f_2 C \sqsubseteq CLS(f_2)$ due to C7.3. Thus $CLS'(f_1) \sqsubseteq$
 $CLS'(f_2).$

ii) $t_2 = f$ and conditions are met. $CLS'(f_1) = CLS(f_1)$
 and $CLS'(f_2) = C$ due to P7.2. From the assumption,
 $CLS(f_1) \leq CLS(f_2)$. From C7.3 and P7.2 we know
 $CLS(f_2) \leq CLS'(f_2) = C$. Thus $CLS'(f_1) \leq CLS'(f_2)$.

iii) Conditions are not met. By P7.2 $CLS'(f_1) = CLS(f_1)$
 and $CLS'(f_2) = CLS(f_2)$. Using the assumption we
 can show $CLS'(f_1) \leq CLS'(f_2)$.

Case 4

All other events.

Px.2 guarantees $CLS'(f_1) = CLS(f_1)$ and $CLS'(f_2) = CLS(f_2)$.

See case 1 i for proof.

Axiom A2.12 For all $e_1 \in E$, $CLR(e_1) \trianglelefteq CLR'(e_1)$.

Case 1 E16 $\underline{e}$ raises its clearance to $\underline{C}$.

Property P16.11 guarantees that either $CLR'(e_1) = CLR(e_1)$ and the axiom is satisfied trivially or $CLR'(e_1) = C$. But C16.1 assures us that $CLR(e_1) \trianglelefteq C$.

Case 2 All other events.

From property Px.11 we know $CLR'(e_1) = CLR(e_1)$ and thus the axiom is satisfied trivially.

Axiom A2.13 For all $f_1 \in F$, $CLS(f_1) \trianglelefteq CLS'(f_1)$.

Case 1 E4 (create $\underline{f}$ in $\underline{d}$).

i) $\neg(f \delta f_1 \wedge C4.1 \wedge \dots \wedge C4.6)$.

From P4.2 we find $CLS'(f_1) = CLS(f_1)$ and thus the axiom is satisfied trivially.

ii) $f \delta f_1 \wedge C4.1 \wedge \dots \wedge C4.6$. ($f = f_1$)

From P4.2 we get $CLS'(f_1) = V.cls$, but C4.5 guarantees that $CLS(f_1) \trianglelefteq V.cls$.

Case 2 E5 (destroy file $\underline{f}$).

In order for $CLS'(f)$ not to be undefined,

$\neg(f \delta f_1 \wedge C5.1 \wedge C5.2)$ must hold. In that case $CLS'(f_1) = CLS(f_1)$ and the axiom is satisfied trivially.

Case 3 E7 (classification of $\underline{f}$ is raised to $\underline{C}$).

There are two possible outcomes for $\underline{f}$'s classification.

i) $\neg(f_1 = f \wedge C7.1 \wedge C7.2 \wedge C7.3)$.

$CLS'(f_1) = CLS(f_1)$ from P7.2 and the axiom holds trivially.

$$ii) \quad f_1 = f \wedge C7.1 \wedge C7.2 \wedge C7.3.$$

From P7.2 we know $CLS'(f_1) = \underline{C}$. But from C7.3,

$$CLS(f_1) \leq C. \quad \text{Thus } CLS(f_1) \leq CLS'(f_1).$$

Case 4.

All other events.

Property Px.2 assures us that $CLS'(f_1) = CLS(f_1)$ and hence the axiom is satisfied trivially.

$$A2.14 \quad \text{For all } m_1 \in M, M\text{-CLS}(m_1) \leq M\text{-CLS}'(m_1).$$

The proofs are similar to those for axiom A2.13 and are omitted here.

$$A2.15 \quad \forall c \in C, c \leq c$$

$$A2.16 \quad \forall c, d, e \in C, c \leq d \wedge d \leq e \quad c \leq e$$

Since the set of security classifications remains unchanged, there is nothing to prove.

A2.17

For all $f_1, f_2 \in F$, $f_1 \delta f_2 \wedge f_2 \in F' \Rightarrow$
 $f_1 \in F'$.

Case 1

E5 (destroy file f)

We know $f_2 \in F'$ or equivalently $ST'(f_2) = \text{USED}$.

But this can be true only if $ST(f_2) = \text{USED}$ and

$\neg(f \delta f_2 \wedge C5.1 \wedge C5.2)$ by P5.1. However

this implies $\neg(f \delta f_1 \wedge C5.1 \wedge C5.2)$ since

if $\neg(f \delta f_2)$ then $\neg(f \delta f_1)$ due to the tran-

sitivity of the " δ " relation on the set of files.

But this along with P5.1 tells us that $ST'(f_1) =$

$ST(f_1)$. This means $ST'(f_1) = \text{USED}$; i.e. $f_1 \in F'$.

Case 2

All other events.

From Px.1 we know $ST(f_1) = \text{USED}$ implies $ST'(f_1) =$

USED .

Appendix E: Mappings and Proofs

In this appendix, we will establish the connection between S_4 and S_3 . First we will present the mapping between levels by showing what the objects and functions of S_4 correspond to in the previous level of specification. Next we will demonstrate (for a representative sample of operations) that the S_4 conditions and properties are sufficient to guarantee that the corresponding event in S_3 can successfully occur. It should be noted that there are cases where the S_3 event could occur but the corresponding S_4 operation would fail. This is because S_4 has more restrictions and hence we will show that the S_4 operation implies the S_3 event. Table E.1 lists the correspondences.

S_4	S_3
CLS	CLS
ACL	Fav
TYPE	ST
RING	Fav
D-CHAR	Fav
L-CHAR	Fav
p-clr	CLR
p-type	Ep
alpar	Ep
childcnt	Ep
alter-attached	α
view-attached	γ
attached	Ep
map	Ep
p-entryno	Ep
branch*	δ
p (process)	e (executor)

Table E.1 Mapping from S_4 to S_3

Since we have given the specifications in terms of functions and the values they return, we will continue to do so. However it should be remembered that these functions correspond to values in repositories and we are in actuality presenting a mapping between repositories with one exception. The status of a file (whether it exists or not) is a physical property and there is no actual repository for each non-existent file to specifically say that it doesn't exist. We assume that the system has some way of knowing that a file exists and returns UNUSED when the status of some non-existent file is requested.

Also, several functions in S_4 may map to a single function in S_3 . For example, "Acl", "Ring", "d-char" and "l-char" all correspond to the S_3 function, "Fav". Thus if in some S_3 event, "Fav" remained unchanged, we would have to show that all of the functions in S_4 which were just mentioned also are unchanged.

An interesting mapping concerns the tree-structure of the file system. In S_3 , the "dominates" relation (δ) is shown to obey axioms A2.5, A2.6, A2.7 and A2.8. These four axioms guarantee that the structure of the file system is a tree*. In S_4 , we have a function called "branch" which when given some integer i , returns the i^{th} offspring of the specified file. Now we say that:

$$\begin{array}{ll}
 f \delta g & \text{iff } g = f \quad (g \text{ is the same as } f) \\
 \text{OR} & \exists n \text{ s.t. } g = \text{branch}(f, n) \quad (n \in \mathbb{N}) \\
 & \quad (g \text{ is one of } f\text{'s offspring}) \\
 \text{OR} & g = f_i \text{ and } f = f_1 \text{ and } \exists n_1, n_2, \dots, n_i \\
 & \text{and } \exists f_1, f_2, \dots, f_i \text{ s.t.} \\
 & \quad f_2 = \text{branch}(f_1, n_1) \\
 & \quad f_3 = \text{branch}(f_2, n_2)
 \end{array}$$

*Actually, the structure is a forest, however the distinction is unimportant.

$$\begin{aligned}f_{i-1} &= \text{branch}(f_{i-2}, n_{i-2}) \\g &= \text{branch}(f_{i-1}, n_{i-1})\end{aligned}$$

In other words "*branch*" can be thought of as the "immediate offspring" relation and δ is the transitive reflexive closure of that relation. Note that substitution and composition can be used with the "*branch*" function so that:

$$f_3 = \text{branch}(f_2, n_2) = \text{branch}(\text{branch}(f_1, n_1), n_2)$$

and

$$f_i = \text{branch}(\text{branch}(\dots(\text{branch}(f_1, n_1), n_2)\dots)n_{i-1})$$

For shorthand we will say:

$$f_i = \text{branch}^i(f_1, N) \text{ where } N \text{ is the sequence } \langle n_1, n_2, \dots, n_i \rangle. \text{ Fur-}$$

ther, we define the transitive reflexive closure on *branch*:

$f_i \in \text{branch}^*(f_1)$ iff $f_1 = f_i$ and $N = \langle \rangle$, or $N \in \mathbb{N}^i$ and $f_i = \text{branch}^i(f_1, N)$ for some $i > 0$. In the proofs to follow we will assume that $f \delta g \iff g \in \text{branch}^*(f)$ and will show that $f \delta' g \iff g \in \text{branch}^{*'}(f)$ that is the equivalence will exist after each operation. One additional point "*branch*" is only defined on files which are in use.

Finally, we can talk about information which the process has. As was detailed earlier in the chapter, each process has information about the file system in the PST. The information contained about a file in the PST is considered accurate if the PST is attached to the file. One of the things we must show is that information in the PST agrees with actuality. Again, we assume correctness before the operation takes place (if the file is attached) and show that the PST is correct after the operation is complete.

Show that S_4 operation #3, GETACCESS (view)

for F

implies S_3 -E1 e becomes view-connected to file f.

$$S_4\text{-C3.3} \quad \text{TYPE}(f) \in \{\text{DATASEGMENT}, \text{DIRECTORY}\} \iff$$

$$S_3\text{-C1.1} \quad \text{ST}(f) = \text{USED}$$

The "TYPE" function in S_4 corresponds to the "ST" or "status" function in S_3 . Since TYPE returns a value different from "UNUSED" it corresponds to the "USED" value for "ST"

$$S_4\text{-P3.12} \implies \text{CLS}(f) \leq p\text{-clr}(p) \iff$$

$$S_3\text{-C1.2} \quad \text{CLS}(f) \leq \text{CLR}(e)$$

The CLS function in S_4 corresponds to the CLS function in S_3 , $p\text{-clr}$ corresponds to CLR and executor e corresponds to process p. Note that an S_4 property is used here to guarantee an S_3 pre-condition. The reason is that "GETACCESS" is more general than "view-connect" since it also corresponds to "alter-connect". However in order to get view access the above property must be true and it is in essence a condition.

$$S_4\text{-C3.1} \quad \text{view-attached}(D) \iff$$

$$S_3\text{-C1.3} \quad e \vee d$$

In the mapping, the function "view-attached" corresponds to the "can-view" relation ($\vee$). Also, D maps to d.

$$S_4\text{-P3.3} \quad \forall f_1, \text{TYPE}'(f_1) = \text{TYPE}(f_1) \iff$$

$$S_3\text{-P1.1} \quad \forall f_1, \text{ST}'(f_1) = \text{ST}(f_1)$$

Since TYPE corresponds to ST, we show that both are unchanged.

$$S_4\text{-P3.1} \quad \forall f_1, \text{CLS}'(f_1) = \text{CLS}(f_1)$$

$$S_3\text{-P1.2} \quad \forall f_1, \text{CLS}'(f_1) = \text{CLS}(f_1)$$

Again, both sides of the mapping are unchanged.

$$\begin{array}{lcl}
 S_4: & & \\
 P3.2 & \forall f_1 & \left\{ \begin{array}{l} ACL'(f_1) = ACL(f_1) \\ RING'(f_1) = RING(f_1) \\ D-CHAR'(f_1) = D-CHAR(f_1) \\ L-CHAR'(f_1) = L-CHAR(f_1) \end{array} \right\} \iff \\
 P3.4 & & \\
 P3.5 & & \\
 P3.6 & & \\
 S_3-P1.3 & \forall f_1, Fav'(f_1) = Fav(f_1) &
 \end{array}$$

Here, all S_4 attributes which map to "Fav" are unchanged and hence we merely show that "Fav" also is unchanged.

$$S_4-P3.15 \quad \forall f_1, \forall n_i, branch'(f_i, n_i) = branch(f_i, n_i)$$

This says that all "branch" functions are unchanged. But we also know that any set of them (in particular those that form a transitive reflexive closure) are all unchanged. Thus $[\forall f_1, \forall N \in \mathbb{N}^i \quad branch^i'(f_1, N) = branch^i(f_1, n)] \iff \forall f \quad branch^{*'}(f) = branch(f) \iff g \in branch^{*'}(f) \iff g \in branch^*(f) \iff S_3-P1.4 \quad \forall f, g, f \delta' g \iff f \delta g$

$$\begin{aligned}
 S_4-P3.12 \quad \wedge \longrightarrow a.alter &\iff \wedge, alter-attached'(F_1) \\
 &= alter-attached(F_1) \iff
 \end{aligned}$$

$$S_3-P1.5 \quad \forall e_1, \forall f_1 \quad e_1 \alpha' f_1 \iff e_1 \alpha f_1$$

For this operation the "alter-attached" function is unchanged which agrees with the fact that the "can-alter" relation is also unchanged.

$$\begin{aligned}
 S_4-P3.12 \quad \forall F_1, view-attached'(F_1) &\iff (a.view \wedge [VIEW-ACCESS \\
 &(f_1, u)] \wedge F_1 = F \wedge C3.1 \wedge C3.2 \wedge C3.3 \wedge [CLS(f_1) \leq \\
 &p-clr(p)]) \vee view-attached(F_1) \implies
 \end{aligned}$$

$$S_3\text{-P1.6} \quad [e_1 \vee f_1] \iff (e_1=e \wedge f_1=f \wedge C1.1 \wedge C1.2 \wedge C1.3) \vee e_1 \vee f_1$$

Several mappings are used for this step. First, if F_1 was previously view-attached then the S_3 "can-view" relation held, and both still hold after the operation/event takes place. The more interesting case occurs when the process did not formerly have view access to the file. Again, we see that in both S_4 and S_3 the file in question is the object of the operation. Furthermore the process (implicitly) and the executor (explicitly) are also the object of the operation. We showed above that the conditions for "GETACCESS" implied the conditions for "view-connect". Thus, we have shown that $S_4\text{-P3.12}$ implies $S_3\text{-P1.5}$. It should be noted that the implication goes only in one direction since $S_4\text{-P3.12}$ has the additional restriction that user u (to whom the process belongs) is on file f_1 's access control list (ACL). The concept of an ACL is unknown at the S_3 level, and hence there is no function to which it maps.

Many of the properties of mailboxes are similar to those for files, and presumably the proofs would be similar. This paper does not cover mailboxes at the S_4 level and hence S_3 properties P1.7-P1.10 are ignored.

$$S_4\text{-P3.8} \quad \forall p_1, p\text{-clr}'(p_1) = p\text{-clr}(p_1) \iff$$

$$S_3\text{-P1.11} \quad CLR'(e_1) = CLR(e_1)$$

S_4 and S_3 guarantee that all processes and corresponding executors do not change clearance. Finally $S_4\text{-P3.9}$ *p-type*, P3.10 *alpar*, P3.11 *childent*, P3.13 *map*, P3.14 *p-entryno* are all made to conform to their respective values in the file system. This and the fact that P3.12 *attached(F)* = TRUE ensure that the values can be taken as correct.

Show that $S_4\text{-#5}$ CREATE FILE $\underline{f}$ as entry $\underline{eno}$ in $\underline{d}$ with attributes $\underline{V}$ implies

$$S_3\text{-E4} \quad \text{executor } e \text{ creates file } \underline{f} \text{ in } \underline{d} \text{ with attributes } \underline{V}$$

$$S_3\text{-P4.1} \quad \forall f_1 \in F, ST(f_1) = \begin{cases} \text{USED if } C4.1 \wedge \dots \wedge C4.6 \wedge f_1 = f \\ ST(f_1) \text{ otherwise} \end{cases}$$

If the S_4 conditions are met, the operation will occur and hence $S_4\text{-P5.15}$ is valid. This is sufficient to guarantee that the S_3 conditions hold. The mappings are obvious.

$$S_4\text{-P5.1} \quad \forall f_1, CLS'(f_1) = \begin{cases} cls(V) & \text{if } C5.1 \wedge C5.2 \wedge \dots \wedge C5.6 \wedge f_1 = f \\ CLS(f_1) & \text{otherwise} \end{cases} \implies$$

$$S_3\text{-P4.2} \quad \forall f_1, CLS'(f_1) = \begin{cases} cls(V) & \text{if } C5.1 \wedge \dots \wedge C5.6 \wedge f_1 = f \\ CLS(f_1) & \text{otherwise} \end{cases}$$

Except for one difference in the mapping, this proof is the same as the last one

$$S_4\text{-P5.2} \quad \forall f_1 \in F, ACL'(f_1) = \begin{cases} acl(V) & \text{if } f_1 = f \wedge C5.1 \wedge \dots \\ ACL(f_1) & \text{otherwise} \end{cases}$$

$$S_4\text{-P5.4} \quad \forall f_1 \in F, RING'(f_1) = \begin{cases} ring(V) & \text{if } f_1 = f \wedge C5.1 \wedge \dots \\ RING(f_1) & \text{otherwise} \end{cases}$$

$$S_4\text{-P5.5} \quad \forall f_1, D\text{-CHAR}'(f_1) = \begin{cases} d\text{-char}(V) & \text{if } f_1 = f \wedge C5.1 \wedge \dots \\ D\text{-CHAR}(f_1) & \text{otherwise} \end{cases}$$

$$S_4\text{-P5.6} \quad \forall f_1, L\text{-CHAR}'(f_1) = \begin{cases} l\text{-char}(V) & \text{if } f_1 = \underline{f} \wedge C5.1 \wedge \dots \\ L\text{-CHAR}(f_1) & \text{otherwise} \end{cases}$$

$\implies$

$$S_3\text{-P4.3} \quad \forall f_1, Fav'(f_1) = \begin{cases} V \text{ if } f_1 = \underline{f} \wedge C4.1 \wedge \dots \\ Fav(f_1) \text{ otherwise} \end{cases}$$

As was pointed out before, Fav in S_3 corresponds to several attributes in S_4 . If we also assume that V in S_3 is appropriately partitioned, then the proof is similar to the two preceding proofs.

We must now show that $S_3\text{-P4.4} \quad f_1 \delta' f_2 \text{ iff } f_1 \delta f_2 \text{ OR } (f_1 \delta d \wedge f_2 = \underline{f} \wedge C4.1 \dots) \text{ OR } (f_1 = f_2 = \underline{f} \wedge C4.1 \dots)$. From $S_4\text{-P5.15}$ we know: $f_2 \in branch^{*1}(f_1) \text{ iff}$

- i) $f_2 \in branch^*(f_1) \quad \text{OR}$
- ii) $d \in branch^*(f_1) \quad (\text{where } N_1 \text{ is the same as } N \text{ except for the final integer in the sequence) and } f_2 = \underline{f} \text{ and C5.1 etc OR}$
- iii) $f_1 = f_2 = \underline{f} \text{ and C4.1 etc.}$

- i) $[f_2 \in branch^*(f_1) \implies f_2 \in branch^{*'}(f_1)] \implies [f_1 \delta f_2 \implies (f_1 \delta' f_2) \wedge C4.1 \dots]$
- ii) $[d \in branch^*(f_1) \wedge f_2 = \underline{f} \implies f_2 \in branch^{*'}(f_1) \implies [f_1 \delta d \wedge f_2 = \underline{f} \implies (f_1 \delta' f_2) \wedge C4.1 \dots]]$
- iii) $[f_1 = f_2 = \underline{f} \implies f_2 \in branch^*(f_1, \{\})] \implies [f_1 = f_2 = \underline{f} \implies f_1 \delta' f_2 \wedge C4.1 \dots]$

We can also show that the S_3 conditions will be satisfied as in previous proofs. Now since $[f_2 \in branch^*(f_1) \iff i, ii \text{ or } iii]$ we know $[f_1 \delta' f_2 \iff f_1 \delta f_2 \text{ OR } (f_1 \delta d \wedge f_2 = \underline{f} \wedge C4.1 \dots) \text{ OR } (f_1 = f_2 = \underline{f} \wedge C4.1 \dots)]$

$$S_4\text{-C5.1: } \text{view-attached}(D) \wedge \text{C5.2 } p\text{-type}(D) = \text{DIRECT}$$

$$\Rightarrow S_3\text{-C4.1 } ST(d) = \text{USED}$$

The fact that D is attached means that 1) we can assume D corresponds to d and 2) we can assume that $p\text{-type}$ is correct; i.e. it is equivalent to $TYPE(d)$. Since $TYPE(d)$ is not $UNUSED$, it agrees with " ST ".

$$\begin{aligned} S_4\text{-P5.15 } \forall n \in N, \forall d_1 \in F \text{ s.t. } d_1 \neq \text{map}(D), \text{branch}'(d_1, n) = \text{branch} \\ (d_1, n). \text{ Hence } \forall n \in N^* \quad \forall d_1 \in \{F-d\}, \text{branch}^{*1}(d_1, N) = \\ \text{branch}^*(d_1, N). \text{ That is } d \text{ is the only file which affects} \\ \text{the structure of the file system. Thus } \exists d_2 \in F \text{ and} \\ \exists n \in N^* \text{ s.t. } \underline{f} = \text{branch}^{*1}(d_2, N) \text{ only if } \underline{f} = \text{branch}^* \\ (d_2, N). \text{ But since C5.3 } \text{type}(f) = \text{UNUSED} \text{ this is im-} \\ \text{possible. Thus we know } \forall f_1 \in F, \forall N \text{ if } f = \text{branch}^{*1}(f_1, N) \\ \text{then } d = \text{branch}^*(f_1, N_1). \text{ This proves } S_3\text{-C4.3 } d \delta f \\ \wedge \forall f_1 \in F, \neg (d \delta' f_1 \wedge f_1 \delta' f \wedge f_1 \neq d) \end{aligned}$$

$$S_4\text{-C5.3 } \text{type}(f) = \text{UNUSED} \Rightarrow S_3\text{-C4.4 } ST(f) = \text{UNUSED. This comes directly from the mapping.}$$

$$S_4\text{-C5.4 } p\text{-cls}(D) \leq \text{cls}(V) \wedge \text{C5.1 } \text{attached}(D) = \text{TRUE} \Rightarrow S_3\text{-C4.5 } \text{CLS}(d) \leq \text{cls}(V)$$

Since D is attached, we can accept $p\text{-cls}(d)$ as correct. The rest is just a matter of mapping.

$$S_4\text{-C5.5 } \text{type}(V) \in \{\text{DATASEGMENT}, \text{DIRECTORY}\}$$

$$S_3\text{-C4.6 } st(V) = \text{USED.}$$

Since DATASEGMENT or DIRECTORY indicate that a file is in use, the above argument specifications correspond.

$$S_4\text{-P5.3 } \forall f_1 \in F, \text{type}'(f_1) = \begin{cases} \text{type}(V) \text{ if } \text{C5.1} \wedge \text{C5.2} \wedge \dots \wedge \text{C5.6} \\ \quad \wedge f_1 = f \\ \text{type}(f_1) \text{ otherwise} \end{cases} \Rightarrow$$

$$S_4\text{-P5.12 } \forall F_1 \in \text{PST } \text{alter-attached}'_f(F_1) = \text{alter-attached}_p(F_1) \iff$$

$$[S_3\text{-P4.5 } \forall e_1 \in E, f_1 \in F \ e_1 \propto f_1 \iff e_1 \propto f_1]$$

Again, this is merely a result of the mapping where $\text{map}(F_1)$ corresponds to f_1 , the processes correspond to executors and "alter-attached" corresponds to " $\propto$ ".

$$[S_4\text{-P5.12 } \forall F_1 \in \text{PST}, \text{view-attached}'(F_1) = \text{view-attached}(F_1)]$$

$$[S_3\text{-P4.6 } \forall e_1 \in E, f_1 \in F, e_1 \gamma' f_1 \iff e_1 \gamma r_1]$$

This is almost identical to the previous proof.

Again, the S_3 mailbox properties, P4.7 - P4.10 are ignored.

$$[S_4\text{-P5.8 } \forall p_1 \in P, p\text{-clr}'(p_1) = p\text{-clr}(p_1)] \iff$$

$$[S_3\text{-P4.11 } \forall e_1 \in E, \text{CLR}'(e_1) = \text{CLR}(e_1)]$$

The mapping is straight forward.

$$S_4\text{-P5.9 } \forall F_1 \in F, p\text{-type}'(F_1) = p\text{-type}(F_1)$$

$$P5.10 \ \forall F_1 \in F, \text{alpar}'(F_1) = \text{alpar}(F_1)$$

$$P5.11 \ \forall F_1 \in F, \text{childcnt}'(F_1) = \text{childcnt}(F_1)$$

$$P5.13 \ \forall F_1 \in F, \text{map}'(F_1) = \text{map}(F_1)$$

$$P5.14 \ \forall F_1 \in F, p\text{-entryno}'(F_1) = p\text{-entryno}(F_1)$$

$$S_3\text{-P4.12 } \forall e_1, \text{Ep}'(e_1) = \text{Ep}(e_1).$$

} $\Rightarrow$

Show that $S_4\text{-\#6}$ DESTROY SUBTREE whose root is $\underline{F}$ implies

$S_3\text{-E}_5$ executor $\underline{e}$ destroys file $\underline{f}$ in directory $\underline{d}$.

$S_4\text{-C6.1}$ $\text{view-attached}(\underline{D}) \wedge$ C6.2 $p\text{-type}(\underline{D}) = \text{DIRECT}$

$\Rightarrow S_3\text{-C5.1}$ $\text{ST}(\underline{d}) = \text{USED}$

Here, the fact that $\underline{D}$ is attached allows us to assume that "p-type" is correct; i.e. it agrees with $\text{TYPE}(\underline{D})$. Again we use the correspondence between "TYPE" and "ST" for the proof.

S_4 -C6.1 $alter-attached(D) \implies e \alpha d$. But due to axiom

A2.3 we get S_3 -C5.2 $CLR(e) \leq CLS(d)$

$$[S_4\text{-C6.1} \wedge C6.2 \wedge P6.3 \forall f_1 \in F, \exists N \in \mathbb{N}^*, f_1 \in branch^*(map(F)) \\ \implies (TYPE'(f_1) = UNDEFINED)] \implies [S_3\text{-P5.1} \forall f_1, f \delta f_1 \wedge C5.1 \wedge \\ C5.2 \implies ST'(f_1) = DELETED]$$

The satisfaction of the S_4 conditions certainly guarantees that the S_3 conditions are true. We have also seen that " $branch^*$ " corresponds to the "dominates" relation (δ). Finally " $TYPE$ " corresponds to " ST " and the value " $UNDEFINED$ " corresponds to " $DELETED$ ". Again we have a case where the S_4 operation is more restrictive than the S_3 event. For example C6.1, $view-attached(D)$, may not happen to be true, though $e \vee d$. However if the S_3 conditions are false, the corresponding S_4 conditions will be false and neither the operation nor the event can take place.

In other words, $\neg S_3 \implies \neg S_4$

$$S_4\text{-P6.1} [\forall f_1 \in F, \exists N \in \mathbb{N}^*, f_1 \in branch^*(map(F)) \wedge C6.1 \wedge C6.2 \implies \\ CLS'(f_1) = UNDEFINED] \implies [S_3\text{-P5.2} \forall f_1 \in F, f \delta f_1 \wedge C5.1 \\ \wedge C5.2 \implies CLS'(f_1) = UNDEFINED]$$

The obvious mapping exists between " CLS " in S_4 and S_3 . The remainder of the argument follows the previous proof. Again, the operation has stronger restrictions than the event.

$$S_4\text{-P6.2} (ACL), P6.4 (RING), P6.5 (D-CHAR), P6.6 (L-CHAR) \implies \\ [S_3\text{-P5.3} \forall f_1 \in F, f \delta f, \wedge C5.1 \wedge C5.2 \implies Fav'(f_1) = UNDEFINED]$$

Without belaboring the issue, we will just state that all S_4 attributes which correspond to S_3 's " Fav " become undefined if conditions are met. Arguments for proof are similar to the previous ones.

$$\begin{aligned}
& [S_4\text{-P6.15 } \forall f_1 \in F, \exists N \in \mathbb{N}^*, f_1 = \text{branch}^*(\text{map}(F)) \wedge C6.1 \wedge C6.2 \\
& \implies \forall n \in \mathbb{N} \text{ branch}'(f_1, n) = \text{UNDEFINED}] \implies [f_2 = \text{branch}^*(f_1, N_1) \\
& \implies f_2 = \text{branch}^*(f_1, N_1) \wedge (C6.1 \wedge C6.2)] \quad (C6.1 \wedge C6.2) \\
& \wedge f_1 \in \text{branch}^*(\text{map}(F)) \implies [f_1 \delta' f_2 \iff \neg(C5.1 \wedge C5.2 \wedge f \delta f_1) \wedge f_1 \delta f_2]
\end{aligned}$$

In other words, if one of the conditions is false, then the "branch*" relation will still hold. If we assume $C6.1 \iff C5.1$, that is if d can be viewed, it will be view-attached, then this implies the $f_1 \delta' f_2$ part of E5. Certainly if C6.1 and C6.2 are true, "branch*" will be undefined. Furthermore, C5.1 and C5.2 will be true and hence $\neg(f_1 \delta f_2)$ for f_2 in the subtree.

$$\begin{aligned}
& [S_4\text{-P6.12 } \forall f_1 \in \text{PST}, \text{alter-attached}'(F_1) = \\
& \left\{ \begin{array}{l} \text{UNDEFINED if } C6.1 \wedge C6.2 \wedge \exists N \in \mathbb{N}^* \text{ s.t. } \text{map}(F_1) = \\ \quad \text{branch}^i(\text{map}(F), N) \\ \text{alter-attached}(F_1) \text{ otherwise} \end{array} \right.
\end{aligned}$$

$$[S_3\text{-P5.5 } e_1 \alpha' f_1 \text{ iff } e_1 \alpha f_1 \wedge \neg(C5.1 \wedge C5.2 \wedge f \delta f_1)]$$

This proof closely follows the previous one.

$$S_4\text{-P6.12 } (\text{alter-attached}) \implies$$

$$S_3\text{-P5.6 } (\text{can-view})$$

This again is too similar to go into more detail.

Show that $S_4\text{-#9}$ Raise Clearance to $\underline{C}$ implies

$$S_3\text{-E16 } \text{Executor } \underline{e} \text{ raises its own clearance to } \underline{C}$$

$$S_4\text{-C9.1 } p\text{-clr} \leq \underline{C} \leq \text{maxclr}(p\text{-user}(p)) \iff$$

$$S_3\text{-C16.1 } \text{CLR}(e) \leq \underline{C} \leq \text{MAXCLR}(e)$$

The mappings are straightforward. Some maximum is assumed to exist for each user.

S_4 -P9.1 - P9.6 unchanged $\implies$

S_3 -P16.1 - P16.3 "ST", "CLS", and "Fav" unchanged.

This is just a simple mapping

$$\begin{aligned} S_4\text{-P9.15} \quad \forall f_1 \in F, \forall n \in N, \text{branch}'(f_1, n) &= \text{branch}(f_1, n) \iff \\ \forall N \in N^i, \text{branch}^i(f_1, N) &= \text{branch}^i(f_1, N) \iff \end{aligned}$$

$$S_3\text{-P16.4} \quad \forall f_1, f_2 \in F \quad f_1 \delta f_2 \iff f_1 \delta' f_2.$$

Again, a simple correspondence.

$$\begin{aligned} S_4\text{-P9.12} \quad \forall F_1 \in \text{PST}, \text{alter-attached}'(F_1) &= \text{alter-attached}(F_1) \wedge \\ (C \leq p\text{-cls}(F_1)) \vee \neg C9.1] \\ \implies [S_3\text{-P16.6} \quad \forall f_1 \in F, \forall e_1 \in E \quad e_1 \alpha' f_1 &\iff e_1 \alpha f_1 \\ \wedge (C \leq \text{CLS}(f_1)) \vee \neg C18.1) \end{aligned}$$

The mapping is fairly straightforward, however one must note that "alter-attached" is more restricted than " α ".

S_4 -P16.12 "view-attached" unchanged $\implies$

S_3 -P16.6 "can-view" unchanged.

$$\begin{aligned} S_4\text{-P9.8} \quad \forall p_1 \in P \text{ (the set of processes), } p\text{-clr}'(p_1) &= \\ \begin{cases} C & \text{if } p_1 = p \wedge C9.1 \\ p\text{-clr}(p) & \text{otherwise} \end{cases} \implies \end{aligned}$$

$$S_3\text{-P16.11} \quad \text{CLR}(e_1) = \begin{cases} C & \text{if } e_1 = e \wedge C16.1 \\ \text{CLR}(e_1) & \text{otherwise} \end{cases}$$

Since the conditions for S_4 and S_3 correspond, both will take on a new classification C under comparable circumstances.

S_4 -P9.9, P9.10, P9.11, P9.13, P9.14 unchanged

$\implies S_3$ -P16.12 E_p unchanged.

Again we have a straightforward mapping.

NOTATION

<u>Symbols</u>	<u>Descriptions</u>	<u>Page Number of Defining Occurrences</u>
S_0	The most abstract Security Structure	11
R	Set of information repositories	14
A	Set of agents	14
C	Set of security classes	14
cls	Classification function for repositories	14
clr	Clearance function for agents	14
$\leq$	Pre-ordering of security classes	14
θ	The "can observe" relation between agents and repositories	14
μ	The "can modify" relation between agents and repositories	14
A0.1	Reflexivity Axiom	16
A0.2	Transitivity Axiom	16
A0.3	Acquisition Axiom	16
A0.4	Dissemination Axiom	16
τ	The "information can pass" relation between repositories	16
τ^*	The "information can eventually pass" relation	17
λ	The "lower classification" relation on repositories	17
Sen	The set of sensitivity levels	19
$Comp$	The set of security compartments	19
C_0	The government Security Lattice	20
T0.1	The basic security theorem	20
S_1	A Security structure with files and mailboxes	27

NOTATION (continued)

<u>Symbols</u>	<u>Descriptions</u>	<u>Page Number of Defining Occurrences</u>
F	The set of files	30
M	The set of mailboxes	30
ρ_F	The "retrieve information" relation for files	30
σ_F	The "store information" relation for files	30
ρ_M	The "receive" relation for mailboxes	30
σ_M	The "send" relation for mailboxes	30
δ	The "dominate" relation on the set of files	30
A1.1-A1.13	The S_1 axioms	31 - 34
S_2	The Structure for Dynamic Security	48
E	The set of executors	48
S	The set of states	48
τ	The transition relation	48
v	The "can view" relation	48
α	The "can alter" relation	48
β	The "can block on" relation	48
ω	The "can wake up" relation	49
E, M, α, cls	Gives the set of executors, mailboxes etc. in each state	50
$E_S, M_S, \alpha, \text{etc.}$	The set of executors, mailboxes etc. current in states	50
A2.1-A2.11	The static axioms for S_2	52
$E_S, M_S, \alpha_S \text{ etc.}$	The sets of transient-executors, transient- mailboxes etc.	63
π	The perpetuates relation	64
A2.12-A2.17	The dynamic axioms for S_2	67 - 68
$S_{1.5}$	The transient-element structure	68

NOTATION (continued)

<u>Symbols</u>	<u>Descriptions</u>	<u>Page Number of Defining Occurrences</u>
S_3	The Dynamic Event Model	74
V_F	set of all possible file attributes values	75
Fav	function which returns V_F , set of file attribute values	75
cls	function to return classification of a file (a particular file attribute value)	75
Pr	set of all properties an executor might have	76
$E1, E2, \text{etc.}$	event numbers	77
$C1.1, C1.2, C10.1, \text{etc.}$	pre-conditions for the event	77
$P1.1, P1.2, P10.1$	properties or effects of a successful event	77
st	function to determine "status" attribute from set of file attribute values	77
ST	shorthand for $Fav \circ st$	77
cls, CLS	return "classification" file attribute	77
clr, CLR	return "clearance" of executor	77
$M-CLS$	returns "classification" attribute of mailbox	78
Ep	returns set of executor properties	78
$V.st$	the "status" parameter	81
$V.status$	the "status" parameter	84
$V.cls$	the "classification" parameter	84
Con	returns the "contents" property i.e. the piece of information being used by the executor	86
$f.info$	some information from file f	86
PST	process segment table	110
CLS	classification attribute	114

NOTATION (continued)

<u>Symbols</u>	<u>Descriptions</u>	<u>Page Number of Defining Occurrences</u>
ACL	access control list	114
CURLN	current length of file	117
DTD	date/time dumped attribute	118
DTM	date/time entry modified attribute	118
DTM	date/time modified	118
DTU	date/time used	119
IACL	initial access control list	119
INFQCNT	inferior quota count attribute	120
SPUSED	space used attribute	120
TACCSW	terminal account switch	120
RECUSED	records used attribute	122
SAFSW	safety switch attribute	122
DID	secondary storage device identifier attribute	122
UID	unique identifier	123
PCLR	process' clearance	123
PUSER	user who owns process	123
ALPAR	alleged parent	124
PENTRYNO	entry number of file with directory kept in PST	125
<i>p-type</i>	returns what executor has listed as a file's type	126
<i>type, TYPE</i>	the "type" attribute of a file	126
<i>branch</i>	function which returns a particular offspring for the specified file.	126
<i>map</i>	returns an internal pointer to a file.	126
<i>p-user</i>	returns the name of the user who owns this process	126

NOTATION (continued)

<u>Symbols</u>	<u>Descriptions</u>	<u>Page Number of Defining Occurrences</u>
<i>p-clr</i>	returns a process' clearance	129
<i>alpar</i>	returns the alleged parent entry in the PST	129
<i>view-attach</i>	boolean function which is true if the specified file is presently view-attached	131
<i>alter-attached</i>	boolean for alter-attached	132
<i>attached</i>	boolean for being attached	132
<i>ACL</i>	returns access control list	136
<i>RLNGB</i>	returns ring brackets	136
<i>D-CHAR</i>	returns the values of all attributes logically located with the file's directory	136
<i>L-CHAR, l-char</i>	returns the value of the local attributes - those located with the file itself	136, 139
<i>V.ring, V.type</i> <i>V.atbs</i>	parameters for the operation	136
<i>spused</i>	function whose value is the amount of storage space used.	139
<i>mayview</i>	boolean function which is TRUE if a given user may view the specified file	141
<i>mayalter</i>	boolean function similar to above but for alter	141