

PROJECT MAC

January 9, 1976

Computer Systems Research Division Request for Comments No. 100

DOMAIN CHANGING MECHANISMS FOR MULTICS

by Warren Montgomery

Several mechanisms have been proposed for the implementation of domains on Multics. This RFC presents these mechanisms and discusses their adequacy for use in controlling process initiation and protected subsystem calling. This RFC does not discuss the technical details of the implementation.

This note is an informal working paper of the Project MAC Computer Systems Research Division. It should not be reproduced without the author's permission, and it should not be referenced in other publications.



Introduction

Protection domains are a very powerful model to use as the basis of a computer protection system. The Multics system places a severe restriction on the use of protection domains. It supports only eight domains per process and forces the access rights of these eight domains to be linearly ordered. These eight domains correspond to the rings of the hardware protection mechanism.

It is not possible for mutually suspicious subsystems to coexist in one process using linearly nested domains. Several mechanisms have been proposed to support more than eight domains per process and to remove the restriction that the domains be nested. We will consider mechanisms for authorizing domain changes. The problem of authorizing domain changes that can occur when a process makes a call is very similar to that of authorizing the creation of a process in its initial domain. Both problems require that the procedures used to enter a particular domain be controlled. A process initiation mechanism, however, must be capable of authorizing domain changes between domains whose access rights are unrelated. A calling mechanism should require that the access rights of the called and calling domains overlap so that references to the arguments of the call can be easily validated. We will describe several mechanisms that have been proposed for controlling domain changes and process initiation. We will evaluate the adequacy of each mechanism for these two problems.

These mechanisms all make use of two new kinds of objects in the Multics hierarchy, domain objects and domain gate objects. This is done because the Access Control List (ACL) mechanism of Multics can be used to perform the actual authorization through the use of the ACLs associated with these objects. Such objects can easily be implemented in a manner similar to the current implementation of message segments. Each of the mechanisms described makes use of both domain gate "objects" and domain "objects". Domain gate objects authorize the domain changes, while domain objects authorize the creation of new domain gate objects. There is a unique identifier for each domain that we will refer to as a Domain identifier (Domain ID). These Domain IDs will be used to designate domains in the same way that Principal Identifiers designate domains in the current Multics implementation. Thus, in the proposed implementation, each Access Control list consists of a list of terms (ACL terms) that specify a Domain ID and a set of accesses. The algorithm used to determine the accesses available to a particular domain is the same as that of the current Multics implementation, unless otherwise specified.

The remainder of this RFC describes four domain changing mechanisms. These mechanisms represent a number of ways to control domain changing in an access control list based system. They represent a number of attempts to obtain the functionality required to support the features of Multics extended to allow mutually suspicious subsystems to execute in the same process.

Included in this set of mechanisms are mechanisms similar to those used by Jones [1] and Schroeder [2] to authorize domain changes.

I have named the four mechanisms to be presented Exact Specification, Partial Specification, Last Component Specification, and Appending Specification. The first two of these mechanisms are intended as process initiation mechanisms, while the last two are for calls of protected subsystems. Exact Specification is the simplest of the four mechanisms, but is not powerful enough to implement the authorization scheme used in Multics. Partial Specification is slightly more complicated, but can easily be used to implement the Multics authorization scheme. Last Component Specification is a mechanism that can be used for protected subsystem calls and is reasonably easily implemented in Multics. Appending Specification is much more general and allows the entire call history of a process to be used for access control. Unfortunately, Appending Specification would be very difficult to implement on Multics.

Exact Specification

The first mechanism for domain entry control to be discussed will be referred to as Exact Specification. Each domain change is authorized by a domain gate object. The domain gate objects each specify a Domain ID and a pathname and entry name. A process makes a call to a procedure in another domain by calling

the "domain call" primitive and passing it the name of a domain gate object. (1) If the process has "call" access to the domain gate object, the domain of the process is switched to that specified by the domain gate and the process executes the specified procedure. To create a process, one must call the process creation primitive passing it the name of a domain gate object to which the caller has "start" access.

The "call" and "start" accesses described above are determined from the ACL of the domain gate. The ACL mechanism is not really needed in this case, as the procedure specified by the gate can perform its own access checking. The important function of the domain gate object is to bind together a procedure and a domain.

The creation of new domain gates is controlled by the domain objects. Each domain object specifies a Domain ID. A process may create a domain gate by specifying a domain object to which it has "create_gates" access and a procedure. The "create_gates" access is determined from the ACL on the domain object. In this case, the ACL mechanism is performing an important protection function.

Domain objects are created by a kernel procedure. This procedure only will create domain objects that specify Domain IDs that have never appeared in domain objects before. We can allow

(1) The dynamic linker could be modified to recognize attempts to transfer to domain gates and automatically perform the call to the "domain call" primitive. The calling procedure could then call the gate just as it would call any procedure in the same domain.

any process in any domain to call this procedure, as creating new domains does not violate security. If desired, however, access to this procedure can be controlled through the ACL on the gate used to call it.

It is important to understand the system of control being employed in this mechanism as it is common to all the mechanisms discussed in this RFC. This system of control is very similar to that used by Schroeder [2] to control the creation and calling of protected subsystems. The creation of new domains is an unprivileged operation, while the creation of gates into a particular domain is under the control of the domain object for that domain.

Notice that access to a domain gate object is sufficient to use a domain gate. Access to a domain object is not required. Thus we cannot, through the ACL of a domain object, revoke the right to use domain gates that were created using that domain object. Adding to the ACL of a domain object is in some sense non-revokable. This non-revocability is true of all of the domain changing mechanisms discussed by this RFC. We could provide some mechanism to destroy all of the domain gates created from a particular domain object. This can be done easily because domain gates cannot be freely transferred or duplicated as could capabilities so that we can keep track of which gates were created from which domain objects.

Exact Specification is sufficiently general to be used for both calling and process initiation in the sense that it can be

used to produce all of the desired domain changes. It also seems relatively straightforward to implement. There are, however, two disadvantages to this mechanism that make it less suitable.

Up to this point, we have assumed that there is one authority responsible for each domain. This is reflected by the fact that there is one ACL that authorizes the creation of new gates into each domain. In the Multics system, there are two authorities that independently authorize entry to a domain. This notion is captured by the multiple component structure of the Principal Identifier. A process executing in a domain with a Principal Identifier of "Jones.CompSys.a" has in some sense been authorized by both Jones and the project administrator for the CompSys project. This is what gives meaning to ACL terms with "*" components, such as "Jones.*.*" . Anyone using such a term knows that all processes that gain access through it have been in some sense authorized by Jones.

In order to preserve this meaning using the Exact Specification mechanism, we cannot allow the domain object generator to generate a Domain ID that matches a previously used Domain ID in any component. If this were not the case, any process could call the kernel to create a new domain object with Jones.New_Project.a as its Domain ID. Gates into this domain could then be created to obtain access to objects through ACL terms of "Jones.*.*" without the authorization of Jones.

The restriction proposed above is too severe in that it makes it impossible to generate Domain IDs that match in some but

not all components. Such Domain IDs would have to be generated by Multics whenever a project has more than one user. The Domain IDs of all users registered on one project will match in the "project" component. We therefore cannot enforce the restriction and still implement the current Multics authorization scheme. The Partial Specification mechanism to be discussed later provides a solution to this problem.

A second major difficulty with this mechanism is that if it is used to control calls, the Domain ID of the called domain is not necessarily related to that of the calling domain. This means that a protected subsystem executing in the called domain cannot make use of the normal access control mechanism to check the caller's access to objects (including the arguments of the call). The calling domain must therefore grant the called domain access to the arguments. The subsystem must be very careful when accessing arguments to be sure that the caller had legitimate access to them. An untrustworthy caller might pass pointers to the subsystem's private data bases and thus trick the subsystem into releasing or modifying information. Partial specification shares this difficulty. Schroeder [2] and Jones [1] present solutions to this problem that fall outside of the usual concept of domains. These solutions involve temporarily augmenting the access rights of the called domain to allow it to get at the arguments. Such solutions can be combined with the domain changing mechanisms presented in this RFC to obtain calling mechanisms adequate for protected subsystem calls, but it is

still useful to try to obtain a domain changing mechanism sufficiently powerful to handle protected subsystem calls without extra machinery. Last Component Specification and Appending Specification are attempts at such mechanisms.

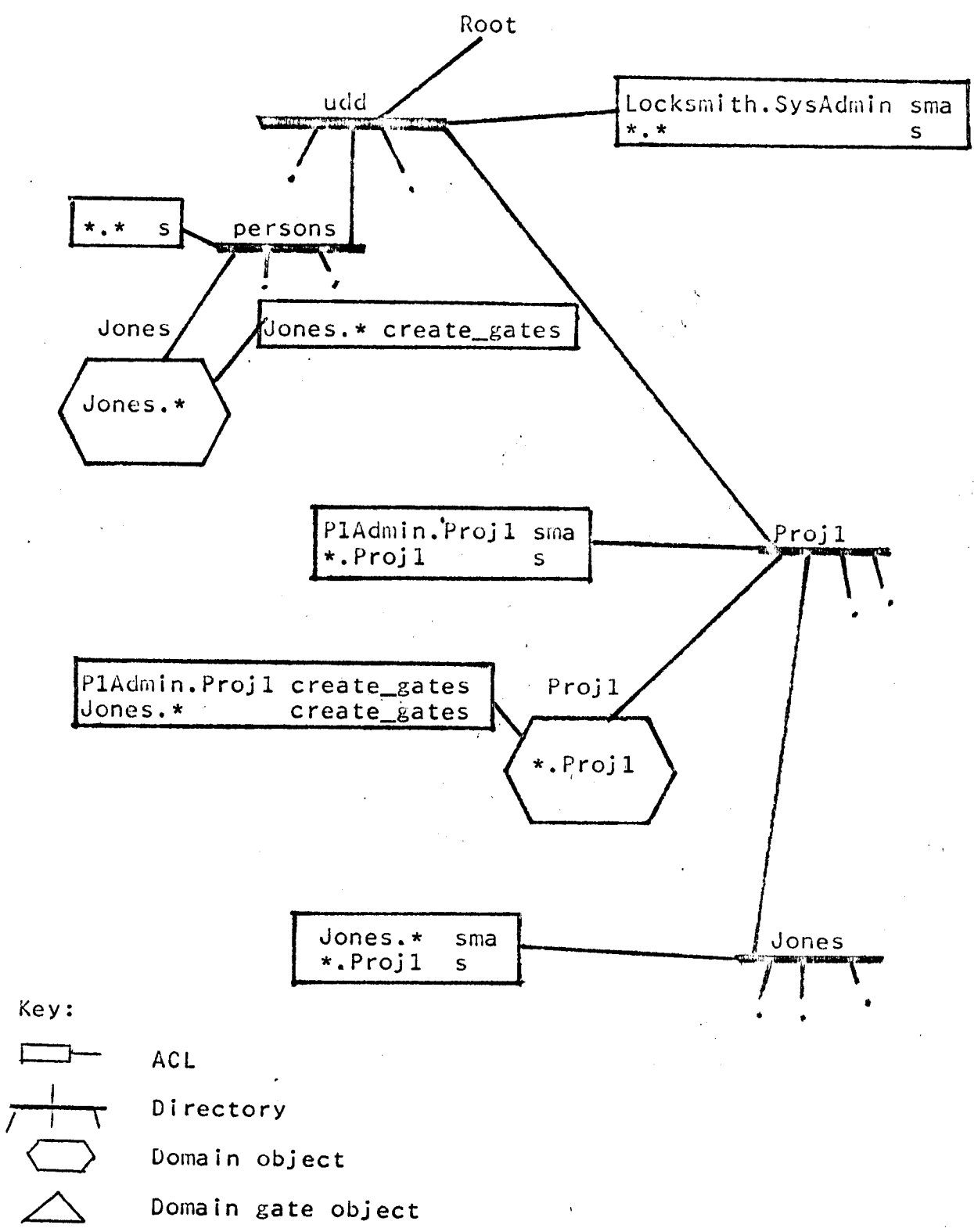
Partial Specification

The second mechanism will be termed here Partial Specification. Domain IDs for this mechanism have a fixed number of components with implied meanings, just as Principal Identifiers have Person, Project, and Instance Tag components in the current Multics implementation. These components represent the independent authorities responsible for each domain. Domain objects in this mechanism specify one or more components of a Domain ID. Domain gates specify a complete Domain ID and a procedure as before. Domain gates are created by passing to a kernel primitive the name of a procedure and a set of names of Domain objects that completely specify a Domain ID. This set must specify each component of a complete Domain ID and not conflict in the specification of any component. For example, Jones.*.*, *.Proj1.*, and *.Proj1.home form such a set, while Jones.Proj2.* and *.Proj1.home do not. Domain gates are used in creating processes and calling subsystems as before. New domain objects that specify previously unused Domain ID components can be created.

Figures 1 and 2 show one way to use this mechanism to implement the current Multics pattern of authorization. Domain

IDs have two components, corresponding to Person and Project. A project is created by creating a domain object specifying only the Project component of a Domain ID. A new user can be registered by creating a domain object that specifies only the Person component. The ACLs on these objects determine who may use them. Notice that the domain Locksmith.SysAdmin is given "sma" access to the directory ">udd". This allows a process executing in this domain to obtain access to any of the objects shown (by modifying ACLs). This domain will have special uses, as shown later.

Figure 1







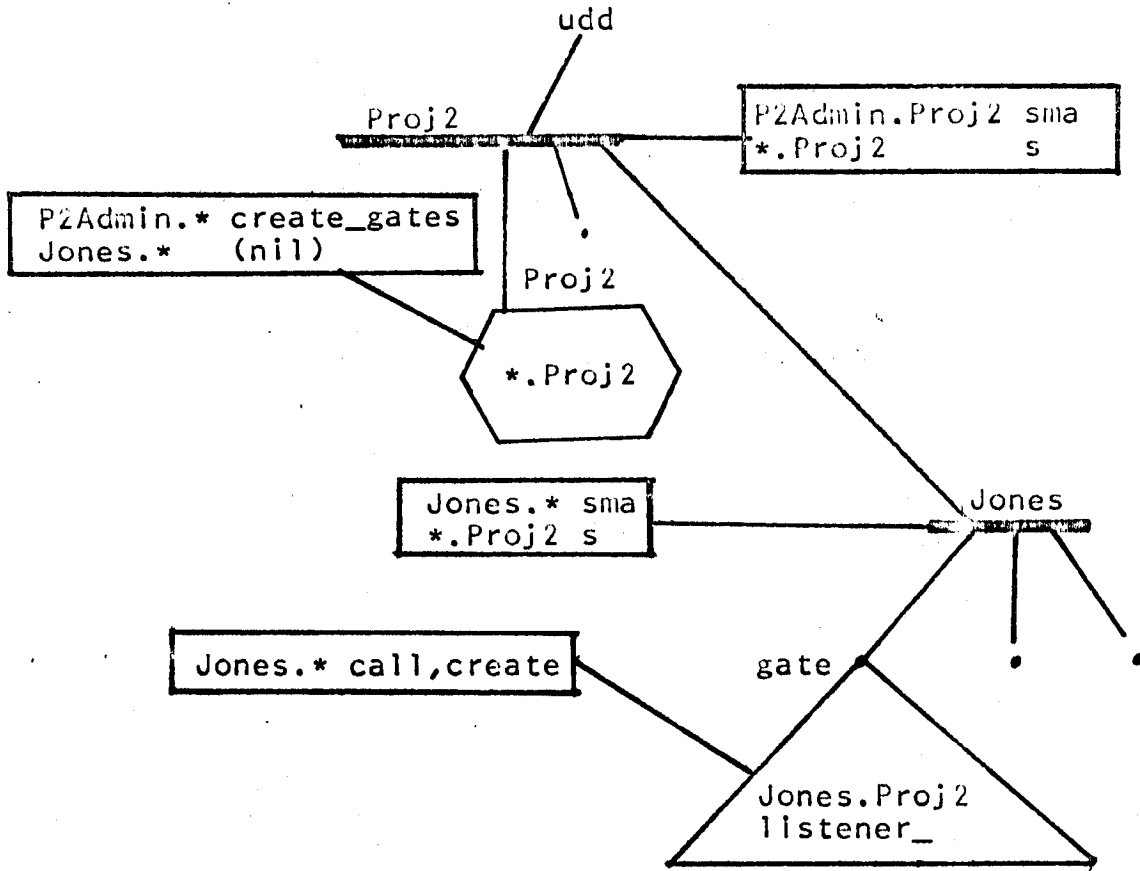
- Key:
-  ACL
 -  Directory
 -  Domain object
 -  Domain gate object

Figure 2



In Figure 1, Jones has been given free access to project Proj1, as he may create new gates into it from any domain with a Domain ID with his name as first component. These gates can be created by presenting the object ">udd>persons>Jones" and the object ">udd>Proj1>Proj1" to the create gate primitive.

Figure 2 shows the hierarchy below the Proj2 directory. Although Jones cannot create new gates into Proj2, he may enter the domain "Jones.Proj2" by using the gate ">udd>Proj2>Jones>gate". This gate had to be created from the domain "Locksmith.SysAdmin", as this is the only Domain ID that can access the domain objects required to create the gate. The procedures of "Locksmith.SysAdmin" would presumably not create such a gate without the approval of both Jones and the administrator for Proj2. The power of the Locksmith.SysAdmin domain should be used carefully. Notice that if at any future time the administrator for Proj2 wishes to allow Jones to create gates to the project, he can do so by modifying the ACL on the object ">udd>Proj3>Proj3", without any help from Locksmith.SysAdmin.

Partial Specification seems to model the Multics authorization mechanism quite well. It does not seem substantially more difficult to implement than Exact Specification and therefore seems to be a more attractive mechanism.

However, this mechanism has the same drawback for subsystem calls as Exact Specification. The calling and called domain are not constrained to share access rights. This requires special action from both caller and callee in order to pass arguments with a call.

Last Component Specification

The third mechanism to be discussed I will call Last Component Specification. This mechanism cannot be used to authorize domain changes between domains whose sets of access rights are unrelated. This makes the mechanism unsuitable for authorizing process initiation, however it makes it more attractive than the previously discussed mechanisms for authorizing protected subsystem calls. As before, Domain IDs have a fixed number of components. Domain and domain gate objects specify only the last of these. (1) A call to a particular gate causes the domain of the calling process to be changed. The domain of the process following the call is that which corresponds to the Domain ID formed by replacing the last component of the Domain ID of the process at the time of the call with the component specified by the gate. Thus if a process executing in the domain "Jones.Proj1.home" made a call to a gate specifying "editor" as its component, the domain of the process would be changed to "Jones.Proj1.editor". New domain objects can be created as before as long as they do not specify the same value as previously created domain objects.

This mechanism is very similar to that proposed in Schroeder's thesis [2] for this purpose. Its implementation in

(1) We could allow them to specify any one component. This generalization does not seem to have any use on the Multics system and just complicates the description of an already complex mechanism.

Multics seems relatively straightforward.

For the Multics system, we could use three component Domain IDs. The first two components of the Domain ID could be used to specify Person and Project, while the last component could specify "Protected Subsystem". All of the subsystems called in a single process will be executed in domains that share some access rights (all access rights that can be obtained by the process through ACL terms with "*" as their third component). Although this does not totally solve the access checking problem discussed before, it does help somewhat by guaranteeing that all of the subsystems in one process will share some access rights.

This mechanism is relatively easy to implement. The component of the Principal Identifier now used for the Instance Tag could be replaced by a Protected Subsystem component without requiring the modification of all ACL terms now in existence. We might want to restrict access to the domain creation primitive to system administrators in order to maintain a list of currently supported subsystems. Each user could then easily find out about a subsystem before placing it on an ACL for one of his objects. The combination of Last Component Specification for calls, and Partial Specification for process initiation, appears to provide a reasonable set of domain changing mechanisms for Multics.

Appending Specification

The last mechanism I will refer to as Appending

Specification. Again, this mechanism is suitable only for calls. The domain and domain gate objects specify only one component of a Domain ID, as in Last Component Specification. The Domain ID of the target domain of a call is formed by appending the component specified by the gate to the Domain ID of the calling domain. A return causes the last component of the Domain ID to be dropped. Thus the Domain ID of a process behaves like a stack, with call and return acting as push and pop instructions.

We can see that Domain IDs can have different numbers of components with this scheme. We therefore need to augment the rules for matching of Domain IDs and ACL terms to specify what happens when the Domain IDs being matched are of different lengths. A reasonable set of rules is the following:

If the Domain ID of the Process is longer than that of the ACL term, then no match can occur.

If the Domain ID of the ACL term is longer than that of the process, then they match only if all of the "extra" components of the ACL term are "*".

In addition to these rules, we will add the rule that a component of "***" in an ACL term specifies exactly enough "*" components to make the ACL term and the process's Domain ID have the same number of components. We will allow only one "***" component in an ACL term. (1) Figure 3 illustrates these matching rules.

(1) Allowing more than one "***" component makes the matching algorithm much more complicated and makes it difficult for a user to see which Domain IDs will match such a term.

Figure 3

Rules for matching ACL terms

ACL term ID	Process Domain ID		
	a.b.c.d	a.b.c	a.b.d
a.**	match	match	match
**.c	no	match	no
a.b.*	no	match	match
a.b.c.d.**	match	no	no

The combination of Appending Specification for subsystem calls and Exact Specification for Process initiation seems to be a very attractive set of mechanisms for authorizing domain changes on Multics. These mechanisms can be used to model the Multics authorization mechanism very well. The first two components of a Domain ID could be used for Person and Project, and the rest would represent the outstanding subsystem calls in the process. All processes would be created in domains with single component Domain IDs (the Person component) and would acquire their Project component through a call. Thus the Person component is authorized through the gate used to create the process and the Project component is authorized through the gate used in the first call. The matching algorithm for Domain IDs allows easy specification of any combination of Person, Project, and current subsystem.

A process can grant access to an object about to be passed by a call by putting a term with the Domain ID of the domain about to be called followed by ".**" on the ACL of the object.

In this way, the object will be accessible to the subsystem to be called and any subsystems that it calls. The ACL term need not be removed following the call, as all of the domains that it matches can only be reached by calling the same subsystem again.

This scheme has a serious practical disadvantage in that the change to the ACL mechanism would be difficult to make, as the length and structure of ACL entries appear in many parts of the system.

With Exact Specification and Partial Specification, each protected subsystem is assigned to one domain. Any call to a particular subsystem always enters the same domain independent of the domain of the caller or the process in which the call is made. With Last Component Specification, the domain that a particular subsystem enters depends on that process it is called in, but not on the subsystem that makes the call. This allows a more precise specification of the access rights to be granted to a particular subsystem. With Appending Specification, the domain in which a protected subsystem executes depends on the subsystem that called it. This allows very precise specification of the access rights to be given to each invocation of a protected subsystem.

There are, however, some undesirable effects of not assigning a particular subsystem to the same domain at each call. As each subsystem can now be invoked in several domains in each process, Appending Specification will tend to use more domains than the other mechanisms. This, and the fact that each domain

will require its own stack and linkage segments, could make this mechanism very expensive. We can also see that linking will have to be performed in every domain that a particular subsystem is invoked.

The "internal static" storage class of PL/1 is now used for variables that are common to every invocation of a particular procedure in a subsystem. With Appending Specification, this will not necessarily be true, as the procedure might execute in different domains in the same subsystem. Although we could change the implementation of Internal Static such that it continues to be accessible to every invocation of a procedure in a subsystem, we would then need a name for storage that was accessible only in one domain. The PL/1 language does not offer enough storage classes to specify all of the classes possible with appending specification.

We might consider placing a limit on the depth of calls to simplify the implementation. In order to allow a reasonable depth of calls, we would have to choose a limit greater than three. This would probably require just as much change to the ACL mechanism as no limit at all.

We might also consider using this mechanism in some limited way to provide independent authorization for Person and Project components. This could be done by allowing a process with a one component Domain ID to make a "call" that appended a second component. Thus a process could be created with only its Person component and could obtain its Project component through such a

"call". This seems significantly more complicated than Partial Specification and does not seem much more effective.

Thus, although Appending Specification appears to be a very powerful mechanism for domain changing, It does not seem suitable for Multics.

Conclusions

This RFC has presented some of the mechanisms proposed for the authorization of domain changing on Multics. The suitability of each mechanism for Multics was considered. The combination of Partial Specification for process initiation and Last Component Specification for calls seems to be the most feasible.

Any comments on these ideas would be greatly appreciated.

References

- [1] Jones, A. K. "Protection in Programmed Systems" Ph.D. Dissertation, Department of Computer Science, Carnegie - Mellon University, 1973.

- [2] Schroeder, M. D. "Cooperation of Mutually Suspicious Subsystems in a Computer Utility", M.I.T. Project MAC, MAC-TR-104, 1972.