PROJECT MAC April 9, 1973

Computer Systems Research Division Request for Comments No. 14

A DESIGN FOR SHARED, USER-PROVIDED, PROTECTED SUBSYSTEMS IN MULTICS

by Richard G. Bratt

Abstract: Multics' current protected subsystem facility is very restrictive. It limits both those who may create and those who may borrow protected subsystems. The attached document presents the functional specifications for a less restrictive protected subsystem facility. This improved facility potentially allows all users to freely create and share protected subsystems. This document assumes the reader has some familiarity with Multics protected subsystems, the problems associated with implementing protected subsystems in Multics, and the current implementation of protected subsystem in Multics.

The ideas presented in this document are part of my undergraduate thesis effort. This effort has been greatly influenced by Professor Mike Schroeder, my thesis advisor. Later documents will cover, in detail, the design outlined in this memo. If time permits, these documents will include a detailed implementation plan. I would greatly appreciate comments on this design as soon as possible.

---

1. Introduction

Multics currently has a very primitive protected subsystem facility. This facility allows each project to define one protected subsystem and share it among its members. Many of the restrictions apparent in this facility are not inherent in Multics' architecture. This memo presents the essential functional specification of a more general, less restrictive, protected subsystem facility.

The design of this new facility is based on the following criterion:

1. The facility must be easy to understand
2. The facility must be easy to use
3. All users must potentially be able to create protected subsystems
4. Users must be able to share their protected subsystems with whomever they choose
5. Changes to the system must be kept to a minimum

2. Defining a Protected Subsystem

We will define a protected subsystem to be a subtree of the Multics hierarchy. The root mode of this subtree is created by a special create_directory primitive. This primitive marks the created directory as being the root node of a protected subsystem. It also records, in the root node, the ring in which the protected subsystem is to execute. The directories and segments in this subtree all need not be part of the protected subsystem. However, all components of the subsystem must be in this subtree.

Protected subsystems may not be nested. This restriction does not seem to eliminate any important cases. More importantly, it eliminates the need for scope rules governing the membership of a segment or directory in an encompassing protected subsystem.

3. Access Control in a Protected Subsystem

In order to protect different protected subsystems from one another and from other procedures, we must be able to specify that access to segments or directories in the protected subsystem is conditional upon executing in that protected subsystem. The obvious way to achieve this conditional access is to allow acl terms that name a protected subsystem. While this solution appears

quite natural, it leads to a fairly complicated facility. Protected subsystem names (tree names) are awkward, dynamic, and non-unique. We feel that this complexity is not warranted and have thus chosen a simpler approach. This approach adds a new mode flag, called "p" for "protected", to the mode field in the access control lists of both directories and segments. This mode specifies that the access, of a process of the principal named in the acl term, to the segment or directory in question is conditional upon that process running in the protected subsystem of which the segment or directory is a member. A more complicated approach would allow one to specify conditional access on a per-mode-bit basis rather than on a per-mode-term basis. While this approach is more flexible, we feel that it adds unwarranted complexity.

The "p" access mode is unlike other access modes in that it grants no specific access rights. Instead, it specifies that the system is to treat this segment or directory as a protected member of its encompassing protected subsystem. Thus, it modifies the effect of the other mode bits. The method by which this protection is effected is quite simple. When a process first attempts to access a protected segment or directory the system verifies that the encompassing protected subsystem is active. If this condition is met then the segment or directory may be accessed, in the ring of execution of the subsystem or in a lower ring, subject to the other mode bits in the acl term and the ring brackets. In cases where "p" access applies the system truncates the R1 and R2 ring numbers of the segment or directory to the ring of execution of the encompassing protected subsystem.

If the encompassing protected subsystem is not active, then all access is denied unless the segment in question is a gate segment. Gate segments are made known to the process, but the process is granted no access to the gate. This forces all instructions referencing the gate segment to fault. If the faulting instruction is a call then the system attempts to activate the encompassing protected subsystem. This process of subsystem activation is discussed later.

We propose that all gates into a protected subsystem be restricted to being directly inferior to the protected subsystem directory. The rationale behind this restriction is that it will make the process of deciding who has access to a protected subsystem easier. It serves no other purpose.

In Multics, a segment is made a gate segment by setting the R3 ring bracket larger than the R2 bracket. Because the security of a protected subsystem depends upon controlling available gates into the occupied ring of a process, it is necessary that a non-null gate bracket be effective only if "p" access to the gate segment applies. (Gates into system ring could be an exception if desired.) Thus, if the matching acl term has the "p" bit on, then the gate bracket is unmodified. However, if the "p" bit is not on then the gate bracket is set equal to the read/execute bracket. The result is that all gates are gates into protected subsystem, and a gate can be used only if the corresponding protected subsystem is active in a process.

4. Invoking a Protected Subsystem

A protected subsystem is invoked by calling a gate segment in the protected subsystem. If the ring in which the gate segment would execute matches the declared ring of execution of the subsystem then the system attempts to latch this ring around the protected subsystem. This latching process is what protects different protected subsystem from one another. A ring, into which a protected subsystem might be latched, may lose its virginity in several ways. It may be a ring reserved for the system (rings 0-2 might be so reserved), it may be the ring that the process logged into, or it may be a latched ring. These are the only ways in which a ring lower then the login ring may lose its virginity, since by definition all gates are gates into protected subsystems (or into the system), and any attempt to invoke a gate from above will latch the corresponding ring. (The next condition eliminates the possibility that a ring higher than the login ring can contain a protected subsystem.) This guaranteed virginity of the latched ring assures that a protected subsystem will execute in a standard, trustworthy, environment.

The second criterion for ring latching is that the ring must be lower than the calling ring and the calling ring must be the lowest latched ring (system rings are not latched, even though occupied.) This guarantees each latched protected subsystem that only protected subsystems which it calls directly or indirectly, or system procedures can be in lower rings. This guarantee is necessary due to the nested access rights of rings. Therefore, each protected subsystem must take care to only call protected subsystems that it trusts.

It should be noted that once a protected subsystem has been latched into a ring, that ring may harbor only that subsystem. Any attempt to call another protected subsystem which executes in the same ring will fail, since the ring is non-virgin. If it is desired to use another protected subsystem which executes in an already latched ring then the old process must be destroyed and a new process must be created. This may be done with the new_proc command.

5. Example

A simple example of how this facility might be used can be abstracted from the problem of controlling access to Multics source code segments. The desired access constraints may be summarized as follows:

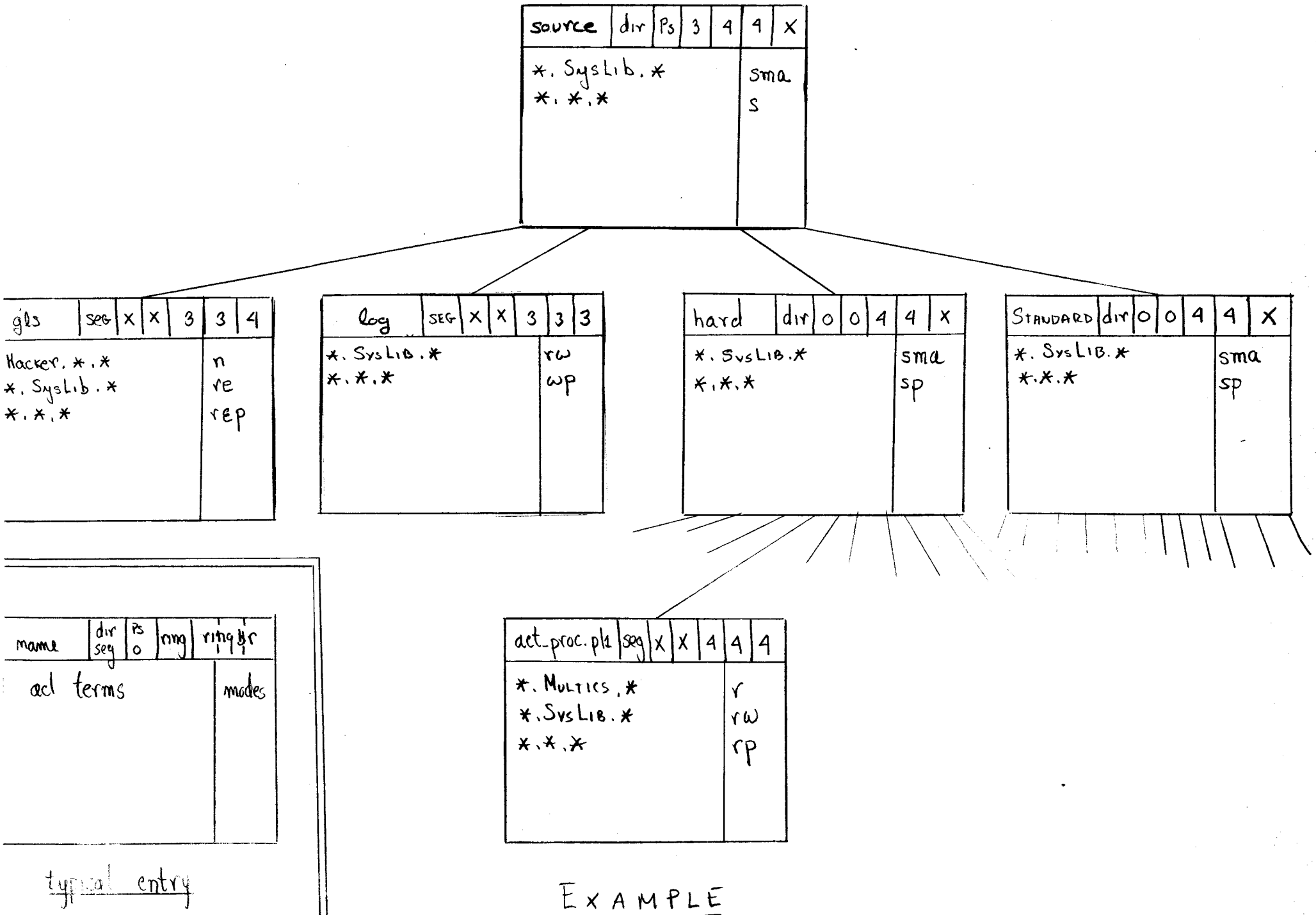
1. Anyone on the SysLib project may read or write all source code segments directly, and list all such segments.
2. Anyone on the Multics project may read all source code segments directly, and list all such segments.
3. All other users are divided into two groups. Those in one group may read source code, but a record should be kept of who reads what and when. Those in the other group should be prevented from reading any source code segments.

We may create a protected subsystem to impose these accessing constraints as follows:

1. login to ring 3 under the SysLib project
2. create a protected subsystem in the directory ">ldd", named "source", to execute in ring 3
3. set the ring brackets on the protected subsystem directory to 4,4
4. give SysLib sma access to the protected subsystem and give everyone else s access
5. create the segment "log" in the protected subsystem directory in which to record accesses to source code (Note that its ring brackets will be 3,3,3).
6. give SysLib rw access to this segment and give everyone else rwp access

7. create a program get_library_source (gls) in the protected subsystems directory which copies source code and records who requested the service in the log segment
8. give SysLib re access, give users who are allowed to read source code rep access, and give those who are not "null" access
9. make the program a gate into ring 3 from ring 4, i.e., set the ring brackets to 3,3,4
10. logout of ring 3
11. login to ring 4
12. copy the source library into a subtree rooted by the protected subsystem directory, giving: SysLib rw access to segments and sma access to directories, Multics r access to segments and s access to directories, and everyone else rp access to segments and sp access to directories.

This scheme allows all users in the SysLib project to directly read and write all segments, and modify all directories in the protected subsystem from ring 4. All users in the Multics project may directly read all source code segments and list their containing directories from ring 4. Finally, all other users either have access to source code only by calling the protected subsystem, or have no access at all. Note that when a user accesses source code through the protected subsystem, the system will truncate the ring brackets of the segment to 3. This prevents the user from touching the segment in ring four. The following diagram represents how this subsystem might look to the system.


source | dir | Ps | 3 | 4 | 4 | X
.SysLib.
..*
sma
s
gls | seg | x | x | 3 | 3 | 4
Hacker.*.*
.SysLib.
..*
n
re
rep
log | SEG | x | x | 3 | 3 | 3
.SysLIB.
..*
rw
wp
hard | dir | 0 | 0 | 4 | 4 | x
.SysLIB.
..*
sma
sp
STANDARD | dir | 0 | 0 | 4 | 4 | X
.SysLIB.
..*
sma
sp
name | dir seg | Ps 0 | rng | ring br | ring br
acl terms
modes
typical entry
act_proc.pl1 | seg | X | X | 4 | 4 | 4
.MULTICS.
.SysLIB.
..*
r
rw
rp
EXAMPLE