On the Synthesis and Analysis of Protection Systems

Lawrence Snyder

Research Report #97

Department of Computer Science
Yale University
New Haven, Connecticut 06520

This work was supported by the Office of Naval Research under grant N00014-75-C-0752.

On the synthesis and analysis of protection systems

Lawrence Snyder

Department of Computer Science
Yale University
10 Hillhouse Avenue
New Haven, Connecticut 06520

Abstract:

The design of a protection system for an operating system is seen to involve satisfying the competing properties of richness and integrity. Achieving both requires the interplay of analysis and synthesis. Using a formal model from the literature, three designs are developed whose integrity (with the help of the model) can be shown.

ON THE DESIGN AND SYNTHESIS OF PROTECTION SYSTEMS

1. *Introduction*

In an enumeration of the many properties that a protection system should have, two distinguish themselves as being especially important:

richness - the property of admitting a complex variety of sharing relationships,

integrity - the property of guaranteeing that the protection system cannot be compromised even in the most hazardous of circumstances.

Both properties are important -- a rich system with dubious integrity is unacceptable, and vice versa. But how are they both to be attained?

It is a difficult task because the two properties are contradicting. For every feature, restriction, exception, etc., added to achieve richness during the *synthesis* phase of design, a complication is introduced into the *analysis* phase of validation. We believe that, traditionally, there has been too much emphasis on synthesis at the expense of analysis. This partly explains why clever systems are so often compromised. It is the purpose of the present report to show how analysis can be used to guide synthesis.

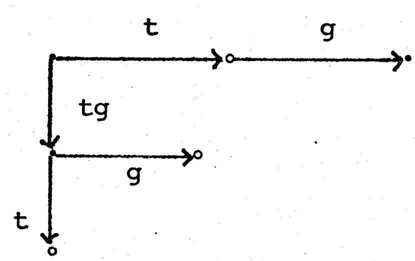
First the theoretical model will be introduced following [1]. The model is graphical and quite intuitive. In [1] the model was studied in some depth and several important properties were proved. These are explained in section 3. Then in section 4 there are presented three basic protection system designs based on the model. Finally, in section 5 the work is discussed together with additional research directions.

2. *Graphical Model of The Take-Grant System*

In this section the Take-Grant protection model of [1] is described (with some modifications*). In order to focus on the role that the model plays in synthesizing protection systems, the Take-Grant system will be presented in purely formal (though quite intuitive) terms in this section and the *interpretation* as a protection model will be postponed until the next section.

The *state* of a Take-Grant Protection system is a directed, edge labelled graph called a *protection graph*. There are two types of vertices in the protection graph, *subjects* and *objects*. (Notationally, filled circles, •, will denote subjects, unfilled circles, ∘, will denote objects, and crossed circles, ⊗, will denote either subjects or objects.) The labels on the edges are called *rights* and are either {t}, {g}, {t,g} where "t" and "g" are mnemonic for "take" and "grant."†

Example 2.1:



A protection graph with three subjects and three objects.

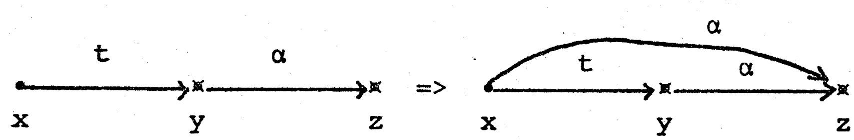
* For those familiar with the model, "t" and "g" labels are used instead of "r" and "w", respectively. The "call" operation has been dropped from consideration and "remove" has been weakened, but not materially.

† We will generally elide the braces around sets.

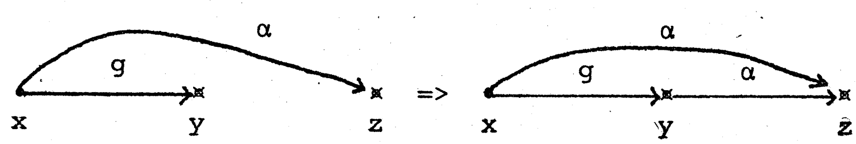
A protection graph G is modified to G' by means of rewriting rules. Rules have the form $\alpha \Rightarrow \beta$. When α matches some subgraph of G, the rule can be *applied* to G, producing a new graph G' (the operation of applying a rule r is written $G \vdash_{r} G'$).

There are four rewriting rules in the Take-Grant Model:

Take: Let x, y, and z be three distinct vertices in a protection graph G such that x is a subject. Let there be an edge from x to y labeled γ such that "t" $\in \gamma$, and an edge from y to z labeled α Then the take rule defines a new graph G' by adding an edge to the protection graph from x to z labeled α.* Graphically,



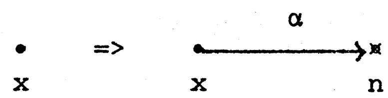
Grant: Let x, y, and z be three distinct vertices in a protection graph G such that x is a subject. Let there be an edge from x to y labeled γ such that "g" $\in \gamma$ and an edge from x to z labeled α. The grant rule defines a new graph G' by adding an edge from y to z labeled α. Graphically,



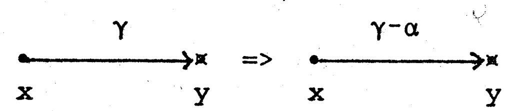
Create: Let x be any subject vertex in a protection graph G and let α be a subset of rights (i.e., α = t, g or tg). Create defines a new graph G' by adding a new vertex n to the graph and an edge from x to n labeled α. Graphically,

---

* In the rules, α is a variable representing any of the three possible labels.

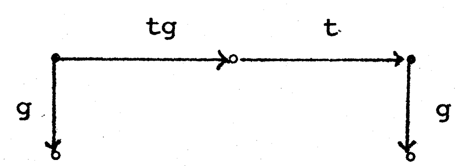


Remove: Let x and y be any distinct vertices in a protection graph G such that x is a subject. Let there be an edge from x to y labeled γ, and let α be any subset of rights. Then remove defines a new graph G' by deleting the α labels from γ. If γ becomes empty as a result, the edge itself is deleted. Graphically,

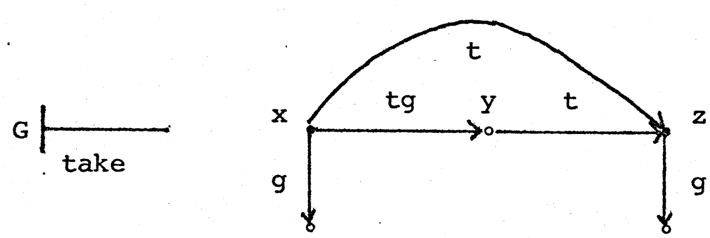


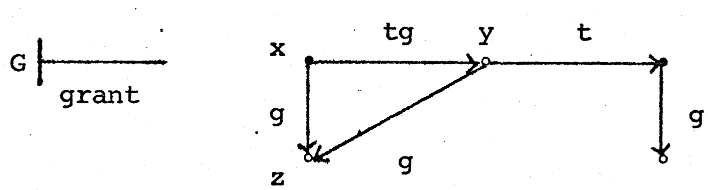
Notice that in the case of take and grant if the edge which is to be added already exists, the label α is simply unioned with the label presently assigned to the edge.

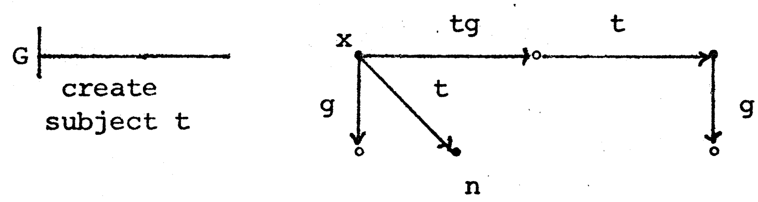
Example 2.2: Let G be

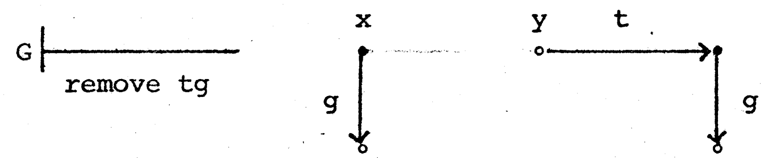


then







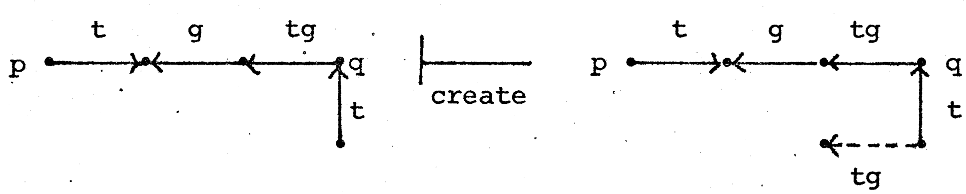


We say that two vertices, p and q, are *connected* if there is path between them without regard to directionality. The vertices p and q are *subject connected* if they are connected by a path whose vertices are only subjects. We say that for vertices p and q of graph G and α a label then *p can* α *q* means that there exists a sequence of graphs $G_0, G_1, \ldots, G_n$ such that $G = G_0 \vdash G_1 \vdash G_2 \vdash \ldots \vdash G_n$ and in G_n there is an edge from p to q with label α.

Theorem 2.1 [1]: Let p, q and r be subject vertices in a protection graph such that there is an edge from r to q labeled α. Then p can α q if p and q are subject connected.

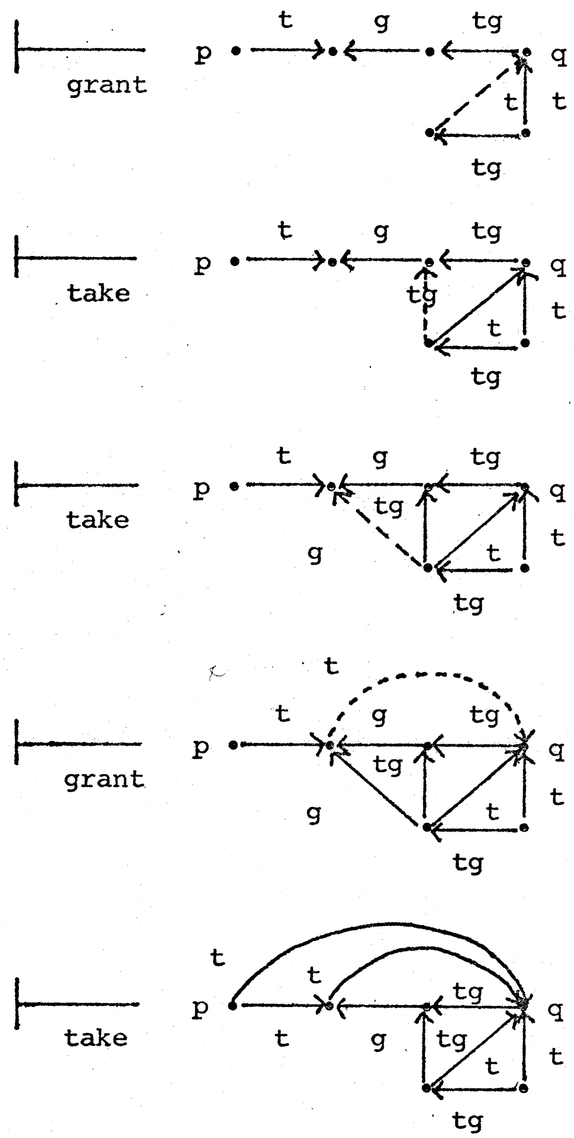
The proof is given in [2], an example should illustrate the result.

Example* 2.3: p can take q



---

* Here dashed lines are used as a visual aid to indicate the added edge.



- The remainder of this section may be skipped on first reading -

A *block* in a protection graph G is any maximal subject connected subgraph. Let p and q be subjects and $x_1,\ldots,x_n$ ($n \geq 1$) be objects such that

p directly connected to x_1

x_i directly connected to x_{i+1}

x_n directly connected to q,

then $p,x_1,x_2,\ldots,x_n,q$ is a *path*. With each such path associate a *word* over the alphabet

$\{\vec{t},\vec{g},\overleftarrow{t},\overleftarrow{g}\}$

letters correspond to edge labels in the obvious way, e.g., $\circ \xrightarrow{\ t\ } \circ$ is represented by $\vec{t}$, and $\bullet \xrightarrow{\ t\ } \circ \xrightarrow{\ t\ } \circ \xleftarrow{\ g\ } \circ \xrightarrow{\ tg\ } \bullet$ is a path associated with the two words $\vec{t}\ \vec{t}\ \overleftarrow{g}\ \vec{t}$ and $\vec{t}\ \vec{t}\ \overleftarrow{g}\ \vec{g}$.

Let E be the union of the regular languages defined by

$$\vec{t}(\vec{t})^{+}$$

$$\overleftarrow{t}(\overleftarrow{t})^{+}$$

$$(\vec{t})^{*}\vec{g}(\overleftarrow{t})^{+}$$

$$(\vec{t})^{*}\overleftarrow{g}(\overleftarrow{t})^{+}$$

$$(\vec{t})^{+}\vec{g}(\overleftarrow{t})^{*}$$

$$(\vec{t})^{+}\overleftarrow{g}(\overleftarrow{t})^{*}$$

where A = AA*. A *bridge* between two blocks exists if from some subject p in one block there is a path with associated word in E to subject q in the other block.

Theorem 2.2 [2]: Let G be a protection graph, p and q and r subjects such that there is an edge from r to q with label α. Then p can α q if and only if there exists a sequence of blocks $B_1,\ldots,B_k$ with p in B_1 and q in B_k and for $i=1,\ldots,i-1$ there is a bridge from B_i to B_{i+1}.

Notice that when k=1, theorem 2.2 strengthens theorem 2.1 to be "if and only if."

3. *Interpretation of the Take-Grant Model*

The development in the last section was presented in graph-theoretic terms and would be valid in any interpretation of letters. Our goal here is to interpret the letters in protection terms.

It is assumed that the protection system is a logically separate entity from the operating system "supervisor" (and thus the supervisor is subject to its limitations like any other process).* In particular, the independence of the protection system allows the user to query the system himself for an *audit* to verify that certain protection conditions hold. The protection graph is a description of the currently extant protection relationships. Thus, the protection relationships among systems entities can be changed *only by the four rules*. The subjects are generally thought to be "user processes" or components that are "active" from a protection point of view, while the objects are thought of as files or processes "known" to be secure. When a subject "applies" a rule (notice that only subjects can "apply" the rules) it is requesting a modification of the protection state. Take causes a user to acquire a new right over some systems entity while grant gives some right away. Create enables new processes and files to have their protection configuration added to the system structure while remove eliminates rights.

Several important facts should be noted about the system:

(3.1) a. take and grant do not create any new rights -- they merely

---

* Here operating system is the totality of the non-user programs while "supervisor" refers to the monitor program.

share existing rights.

b. rights, once removed, can never again be restored.

c. rights, once granted, can never be recovered (i.e., once rights are granted away, they can be distributed by the recipient without consulting the grantor).

In addition to these obvious properties of the model, the two theorems give further information about what is possible in the Take-Grant systems.

Specifically, theorem 2.1 can be interpreted as saying:

> *"Given a collection of users that are connected, if some user has a particular right over another user, then every user can acquire that right."*

This result suggests, but by no means *proves* (see below), that the Take-Grant system is very weak. After all, how can there be any sharing among users if everyone can potentially get the objects that one user intended for another? To be safe it appears that users must be unconnected. Moreover, the second theorem does not give much hope, since it implies that in order to "buffer" against some unwanted security leaks there must be at least two objects separating the various user blocks. But once again it is not possible to share without the potential of having eveyone acquiring the rights. The Take-Grant System may be an analyzable system, but it doesn't appear to be rich!

It would be premature to dismiss the system as being too weak. The theorems indicate what *can* happen and what *cannot* happen. In the former case, the *proofs* of the theorems tell how various rights can be acquired when they can be and this is the key to designing a richer system than would appear possible. This will be shown in the next section. With the analysis at hand, it is possible to know the consequences on system integrity of design choices.

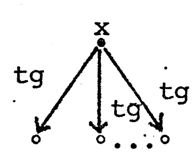
4. *Take-Grant Systems Designs*

In this section, three designs will be presented based on the Take-Grant model. The focus is on understanding how rich each design is (i.e., what information is protected and what is exposed) by employing the analysis of section 2 together with the interpretation of section 3.

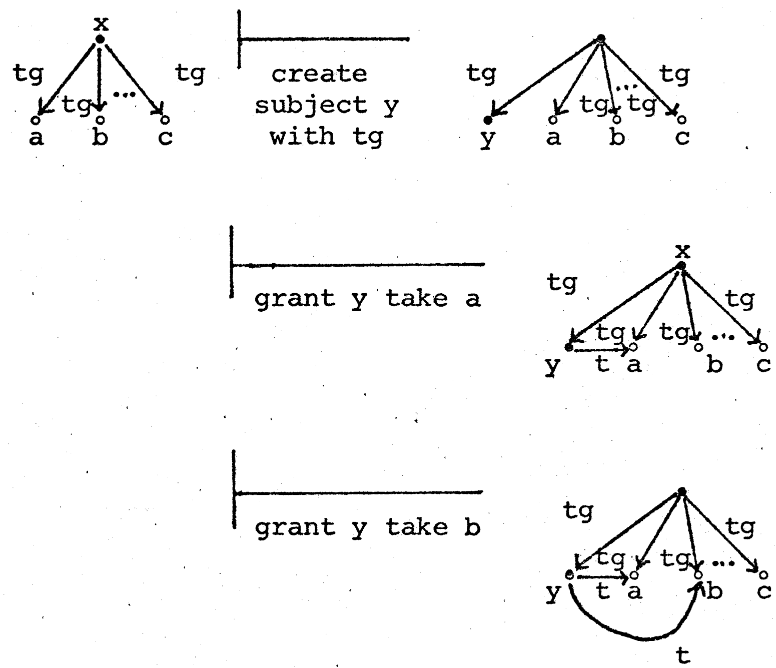
As indicated in the last section, the operating system supervisor is distinct from the protection system and is thus treated just like any other subject in the system. Of course, it does have a special role of joining new users to the system, managing library programs, etc., so considerable interest will be directed toward understanding how it might perform these functions. Accordingly, the initial configuration and the protocol followed by the operating system will be of crucial importance.

4.1 *General form of user processes*

Normally, a user x will be described by the protection subgraph

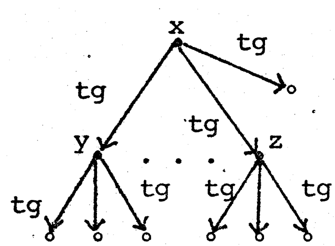


assuming that no sharing is currently active. Here x is the user and the objects are files. To create a subprocess y to operate on two files a and b, the user simply performs the protection functions

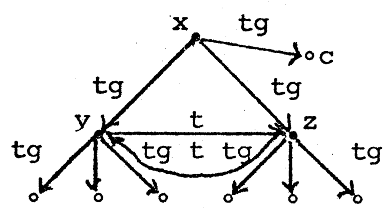


Such a user is called a *greedy* user, since he does not share.

A second general user form achievable in the model are the *project* users, used, for example, by a group jointly writing a compiler. Here x is the project leader (created by the system) while y and z are project workers (created by the project leader) and the graphical representation is



where y and z have created their own files, as does x. Of course, y and z's files should be generally available to all who are working on the project, and the leader enables mutual access by granting y and z take rights over each others' files.



With a take y can access z's files and vice versa. Other general user structures can obviously be envisioned, e.g., instructor - teaching assistant - students, and the reader is invited to design them.

4.2 *Theft in the take-grant system*

Notice that according to theorem 2.2 y *can* take rights to c in both the greedy user and project user structures. Does this mean that y can take control of a file that x wants to keep secure? Emphatically not! The reason is that *y cannot take control of c without x giving the rights away.* Hence, if x wishes to keep it secure, x can choose to do so. This distinction between what *can* happen and what might reasonable take place is absolutely crucial to assessing the utility of the take-grant system. It can be summarized as follows:

> *Theorem 2.2 defines exactly the protection relations achievable in an* arbitrary *state by means (if necessary) of the combined effect of all system subjects. A maximally rich design with the integrity property restricts the achievable relations to those in which the creator of the information* must *participate in its dissemination.*

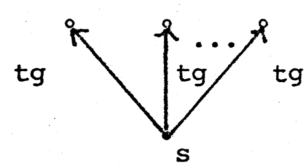
With this distinction in mind, various systems designs may now be considered. We avoid creating arbitrary states and focus instead on "controlling" system growth.

The designs depend on a simple fact of the Take-Grant Model:

> If x is a subject, x has no incoming edge labeled "t" of "tg", and if the rights to any subject or object created by x can be acquired by some other subject or object y, then y can acquire the rights only if x (initially) grants the rights away.

Thus, subjects satisfying the "no incoming take" requirement can control what they create. The "initially" caveat is necessary by 3.1c since once control is relinquished anything *may* happen.

In each design the operating system supervisor is the initial subjsct in the system together with its "service objects", i.e., library files, etc. Thus each of the following systems has as its initial configuration



where s is the operating system supervisor and the objects are the "service objects." *Notice that no edges are incoming for s, so none will ever be introduced* (by theorem 2.2), so no user will be able to take from the supervisor.

4.3 Model 1 - Operating system as communications agent

In this design the supervisor communicates with the systems users by means of an object (thought of, possibly, as a buffer). The users communicate with one another by requesting the operating system supervisor to act as intermediary.

The protocol for introducing a new user x to the system is:

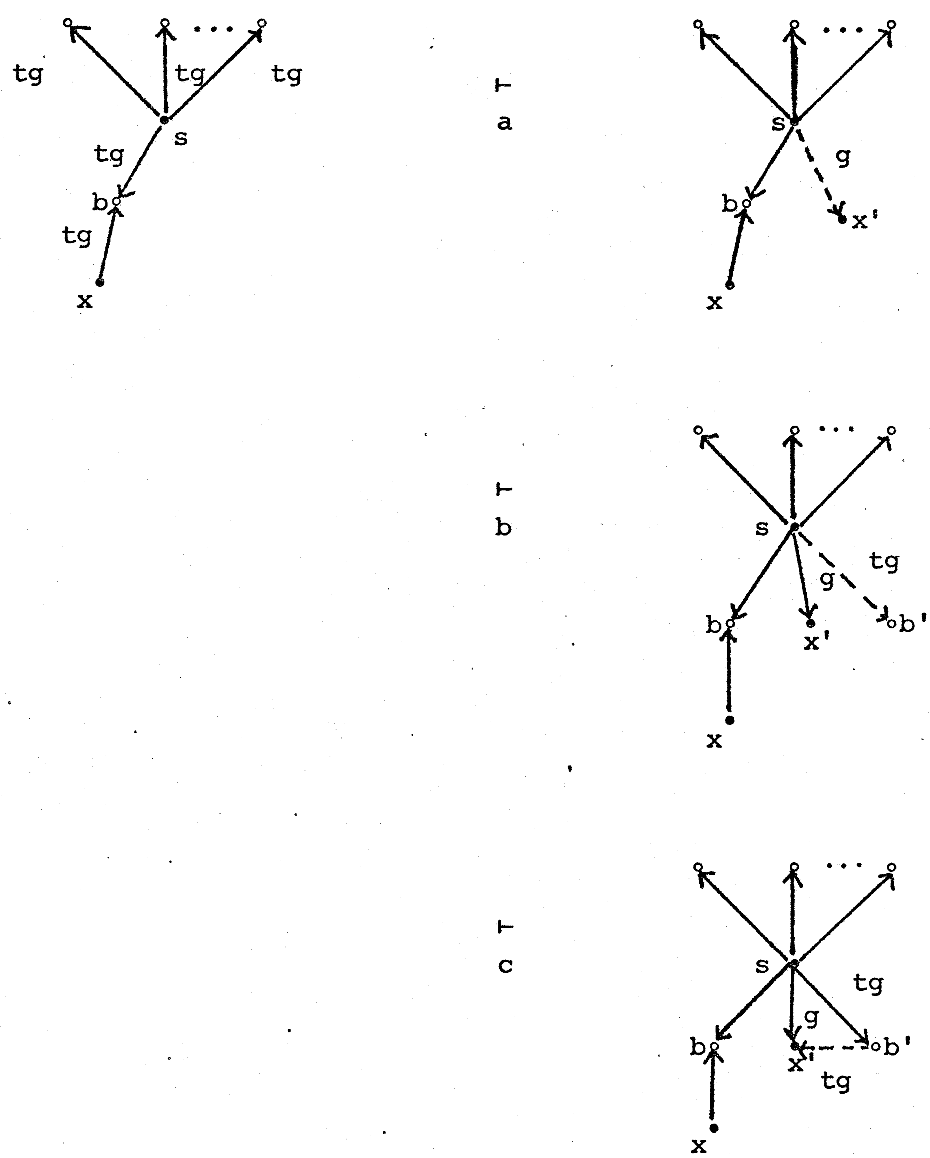
a. create subject x with g

b. create object b with tg -- this is the buffer

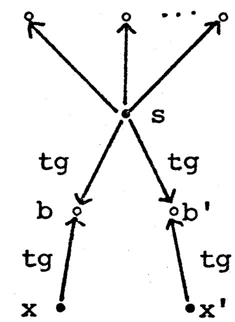
c. grant x tg to b

d. delete g from x.

Graphically, a system with one user, x, can have a new user x' added as follws:

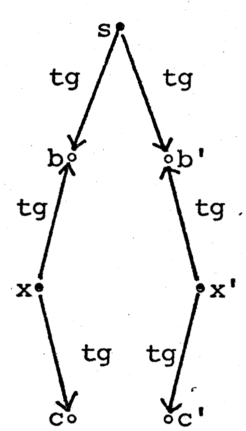


$\vdash$
d



Notice that the user must *trust* the supervisor not to perform step (a) with grant *and* take and then to retain the take right since this would enable the supervisor to take anything created by the user. But if the user requests an audit from the protection system as its first act of business, it can be verified that no such rights exist. Notice that no arrows are incoming to a user so it can establish a subsystem with the same features as the overall system -- i.e., the user acts as supervisor to its subordinates.

Given the configuration (when the service objects have been elided)



x can be given rights to c' using the following protocols.

	subject x	subject s	subject x'
a	create object d with tg		

	subject x	subject s	subject x'
b	grant "tg" to d to b		
c		take "tg" to d from b	
d		grant "tg" to d to b'	
e		delete "tg" to d	take "tg" to d from b'
f			grant "t" to c' to d
g	take "t" to c' from d		

Here d acts as a receptical for the data.

In step e the operating system yields its right to possibly taking the data and prior to step f, a paranoid x' could request an audit to verify that s yields its rights *and* that the others have followed the protocol.

Whether or not this design is adequate is dependent on the system's requirements -- a question that cannot be answered here. However, it should be noted that with the supervisor as intermediary there could be a lot of traffic. Thus, in an effort to reduce this, a second design is considered.

4.4 *Model 2 - No agent*

Here the operating system supervisor sets up a buffer (such as b in Model 1) between each user pair. Then the sharing responsibilities are placed on the users rather than the supervisor. In addition, the supervisor must retain grant rights over all of the users in order to establish the communication.

The protocol for introducing new users assuming $x_1,\ldots,x_n$ already exists is:

a. create subject y with "g"

b. create object b_1 with "tg"

c. grant "tg" to b_1 to y

d. grant "tg" to b_1 to x_1

e. delete "tg" to b_1

. . . .

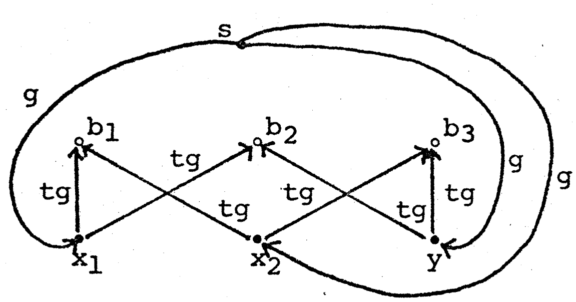
f. create object b_n with "tg"

g. grant "tg" to b_n to y

h. grant "tg" to b_n to x_n

i. delete "tg" to b_n

The following configuration results when y is added and x_1 and x_2 exist.



Communication among users is a simple task and is left as an exercise.

The design may reduce the variable cost by eliminating communication traffic, but it raises the overhead of the supervisor to be proportional to the number of users in the system. Moreover, the protection system is swamped with information. If modest sharing among processes is anticipated, model 3 might be preferred.

4.5 *Model 3* - *The supervisor as communications linkage agent*

The obvious solution to the shortcomings of Models 1 and 2 is to combine the features -- i.e., the supervisor sets up communication buffers on demand. Thus, the supervisor's work is proportional to the number of users sharing rather than the amount of sharing. Also, only those links that are needed are created.

The user creation protocol for user x is simply

a. create subject x with "g"

when sharing between subjects x and y is required, the protocol for the supervisor is

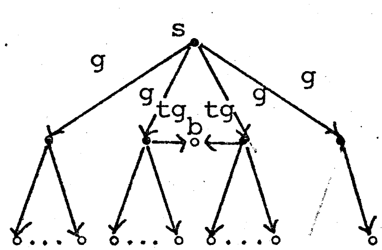
a. create object b with "tg"

b. grant "tg" to be to x

c. grant "tg" to b to y

d. delete "tg" to b.

A sample configuration among four users with two of them sharing might be:



The communication protocol for the users is obvious. Notice also that the users might request an audit once the object b has been created. Moreover, in this scheme (and in the other models as well) any user can decide to isolate himself simply by performing delete. But by 3.1b, he does so in model 1 at the risk of perpetual isolation.

5. *Discussion*

The three models in the last section do not exhaust the possible designs, nor do they represent necessarily good designs. The appropriateness of any particular design is contingent on the system's requirements and these are for the designer to assess. They do show some alternatives with a certain degree of richness.

The point to be emphasized, however, is that the formal Take Grant Model provides a means of guiding the synthesis of a design and it enables analysis of the result. For example, in the forgoing models no user is ever allowed by the operating systems supervisor to have an incoming edge labeled by t since this would allow the potential of having rights taken without the user's participation. Should a user decide that it desires such rights over its own subsystems, (i.e. the ability to steal), then it can create them in this manner. If it is less interventionist than that it could create subsystems after models 1-3. In any case the fact that the system has been analyzed and characterized enables everyone to know the potential consequences of their actions.

Finally, it should be noted that the Take-Grant Model is not necessary being advocated here, although it does appear to be useful. What is being advocated is the use of some formal model in which information such as that embodied in Theorems 2.1 and 2.2 is known. This seems the only possible way to achieve integrity.

Accordingly, as future research directions the following can be suggested:

- Assess the designs discribed here from a richness and an efficiency of implementation viewpoint
- Find alternative designs within the Take Grant Model to achieve even greater richness
- Find extensions to the Take Grant Model which are more expressive with a greater number of rights and/or rules.
- Find alternatives to the Take Grant Model to remedy problems not curable in the forgoing approaches.

Reference

1. A. K. Jones, R. J. Lipton and L. Snyder. A linear time algorithm for deciding security. Proceedings of the 17th FOCS (1976).