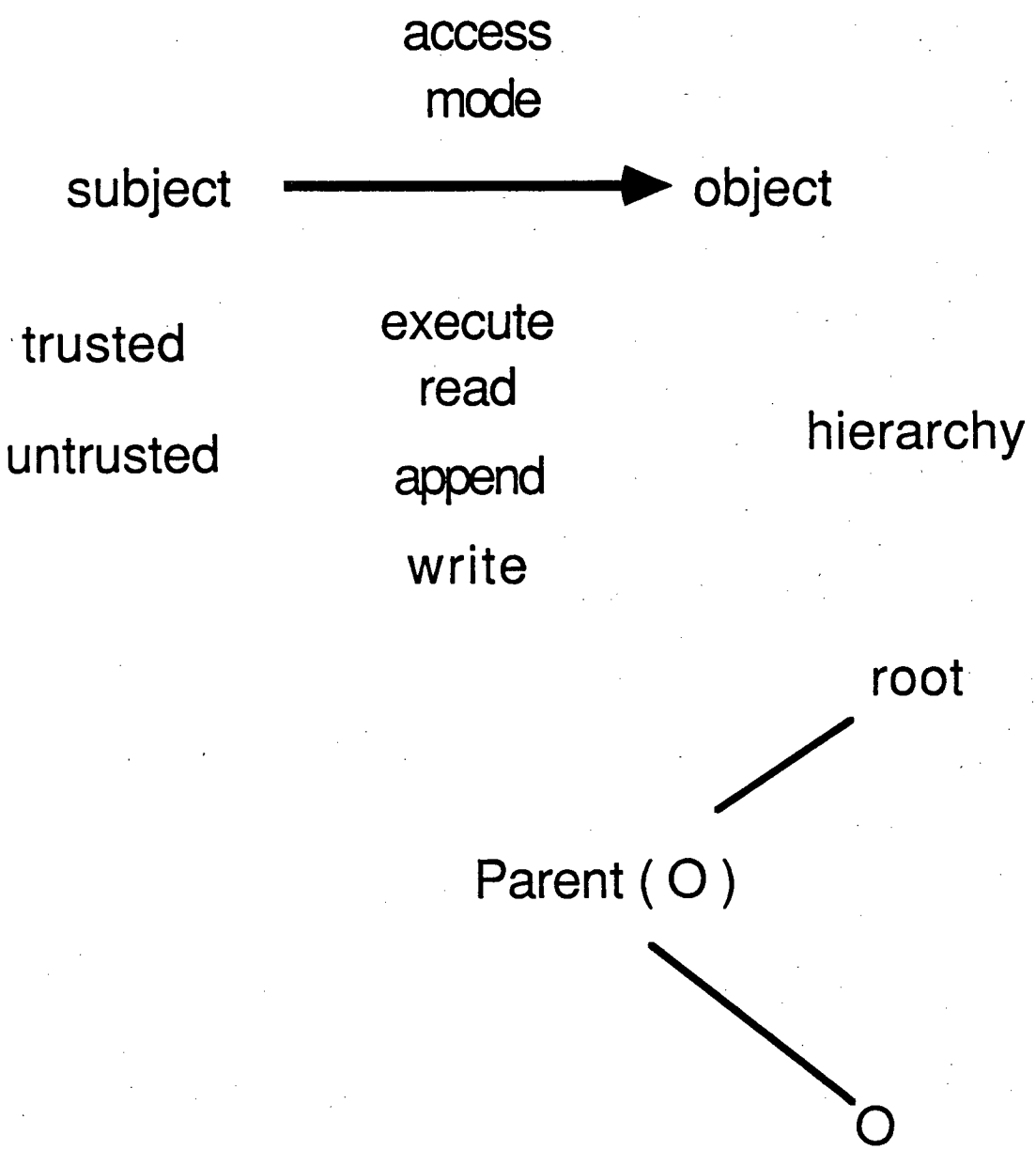
BELL - LA PADULA SECURITY MODEL

B1 - LEVEL SECURE UNIX FILE SYSTEM

BELL - LA PADJULA MODEL

- perspective
- constructs
- problems


access
mode
subject
object
trusted
untrusted
execute
read
append
write
hierarchy
root
Parent (O)
O

	O1	O2		
S1	R			
s2				

current

access

matrix

	O1	O2		
S1	R W			
s2		E		

permissions

matrix

Security Labels (Levels)

subjects : current level, maxlevel

objects : (current) level

DOMINATES relation

1. L1 >= L1

2. L1 >= L2 and L2 >= L1

   implies L1 = L2

3. L1 >= L2 and L2 >= L3

   implies L1 >= L3

```
$ TYPE DECLARATIONS
$ Entity includes both subjects and objects.

TYPE Entity;
TYPE Subject OF Entity;
TYPE Object OF Entity;

TYPE SecurityLevel;

$ Declare access modes.

TYPE Access = (e_access, r_access, a_access, w_access);

$ Declare access elements.

TYPE Elt;                  $ Declare access elements as a type.
TYPE EltSet;               $ We will also need sets of access elements.

$ SET MACROS

$ Use the SetDefs macro to the pull in the definitions of
$ set operations for sets of elements.

SetDefs( Elt, EltSet )

$ This is part of the output of the SetDefs macro applied to elements:

$ Declare "is a member of" primitive for sets of elements.

DECLARE InEltSet(Elt, EltSet) : BOOLEAN;

$ Axiom of set union.

DEFINE  UnionEltSet(set1EltSet, set2EltSet) : EltSet
BY
    FOR elt1Elt
      EQUIV
      {
        InEltSet(elt1Elt, UnionEltSet(set1EltSet, set2EltSet));
        OR
        {
          InEltSet(elt1Elt, set1EltSet);
          InEltSet(elt1Elt, set2EltSet);
        };
      };
$ Besides the basic operations, axioms for set equality and the
$ empty set are generated.
```

```
$ MORE TYPES

$ Each Elt is a triple, (Subject, Object, Attribute):

DECLARE SubjectOf( Elt ) : Subject;
DECLARE ObjectOf( Elt ) : Object;
DECLARE AttributeOf( Elt ) : Access;

$ Declare the current access set.

DECLARE CurrentAccess( Time ) : EltSet;

$ Declare the access permission matrix as a function
$ of subjects and objects.

DECLARE Permissions( Time, Subject, Object ) : AccessSet;

$ Declare security level functions.

DECLARE MaxLevel( Subject ) : SecurityLevel; $ maximum level
DECLARE ObjectLevel( Time, Object ) : SecurityLevel;
DECLARE CurrentLevel( Time, Subject ) : SecurityLevel;  $ current level

$ Declare the "dominates" relation for security classes.

DECLARE Dominates( SecurityLevel, SecurityLevel ) : BOOLEAN;

$ Define some nctions associated with the hierarchy of objects.

DECLARE Parent( Time, Object, Object ) : BOOLEAN;

VAR obj, obj1, obj2 : Object;

DEFINE Root( t, obj ) : BOOLEAN
BY
  FOR obj1 NOT Parent( t, obj1, obj );

$ This is a recursive definition of obj1 being inferior to obj2:

DEFINE Inferior( t, obj1, obj2 ) : BOOLEAN
BY
  OR
  {
    Parent( t, obj2, obj1 );
    EXIST obj
     AND
     {
       Inferior( t, obj, obj2 );
       Parent( t, obj, obj1 );
     };
  };
```

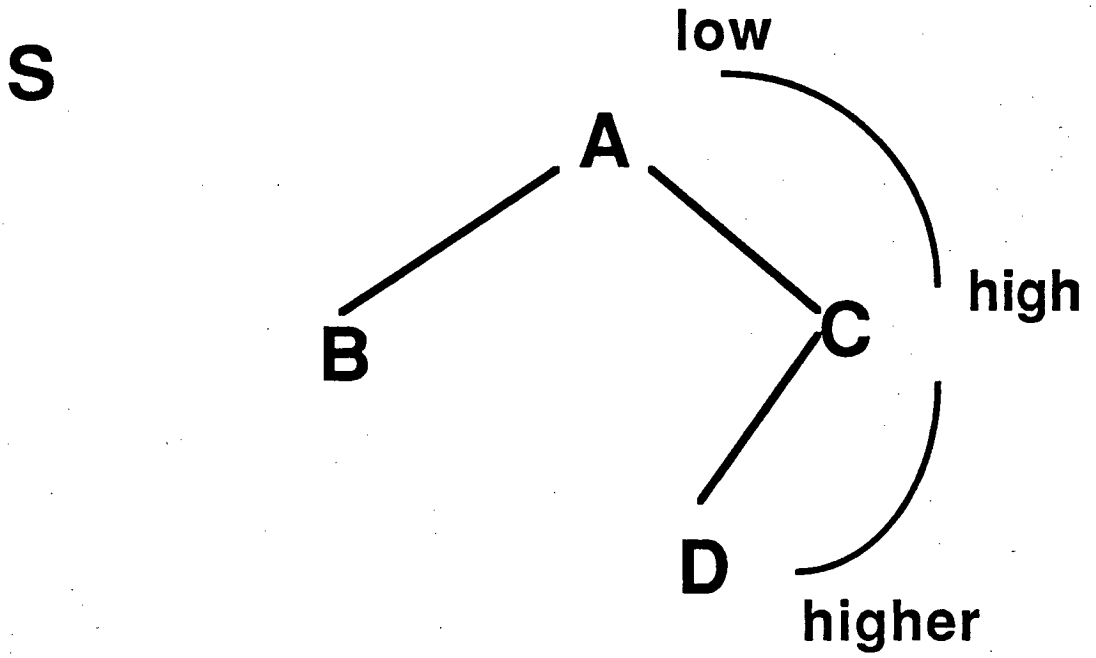
SECURITY POLICY

- simple security (applies to all)
- star - property (only untrusted)
- discretionary control (applies to all)

```
$ STATE REQUIREMENT

$ Declare the security properties of the system.

DECLARE SimpleSecurity( Time ) : BOOLEAN;
DECLARE StarProperty( Time ) : BOOLEAN;
DECLARE DiscretionarySecurity(Time) : BOOLEAN;

DEFINE StateRequirement( t ) : BOOLEAN
BY
  AND
  {
    SimpleSecurity(t);
    StarProperty(t);
    DiscretionarySecurity(t);
  };

$
$ Define the simple security condition.
$ Note: InEltSet is defined in SetDefs.l.
$ It is the relationship of set membership between
$ access elements and sets of access elements.
$
DEFINE SimpleSecurity(t) : BOOLEAN
BY
  FOR elt
    IF AND
    {
      InEltSet( elt, CurrentAccess(t) );
      ObserveAccess( elt );
    }
    THEN Dominates( MaxLevel(SubjectOf(elt)),
                    ObjectLevel( t, ObjectOf(elt)));
```

```
$ Define the *-property.

$ If Trusted (subj) Then subj is a trusted subject.

DECLARE Trusted( Subject ) : BOOLEAN;

DEFINE StarProperty(t) : BOOLEAN
BY
  FOR elt
    IF AND
    {
      InEltSet( elt, CurrentAccess(t) );
      NOT Trusted( SubjectOf(elt));
    }
    THEN AND
    {
      IF AttributeOf(elt) = a_access
      THEN Dominates( ObjectLevel(t,ObjectOf(elt)),
                      CurrentLevel(t,SubjectOf(elt)));

      IF AttributeOf(elt) = w_access
      THEN ObjectLevel(t,ObjectOf(elt)) = CurrentLevel(t,SubjectOf(elt));

      IF AttributeOf(elt) = r_access
      THEN Dominates( CurrentLevel(t,SubjectOf(elt)),
                      ObjectLevel(t,ObjectOf(elt)));
    };

$ Define discretionary security.
$ Note: InAccessSet is similar to InEltSet except that it denotes
$ set membership between access attributes and sets of access
$ attributes.

DEFINE DiscretionarySecurity(t) : BOOLEAN
BY
  FOR elt
    IF InEltSet( elt, CurrentAccess(t) )
    THEN InAccessSet( AttributeOf( elt ),
                      Permissions( t, SubjectOf(elt), ObjectOf(elt)));
```

COMPATABILITY PRINCIPLE



level (S) >= level (A) read A

level (S) <= level (A) write A

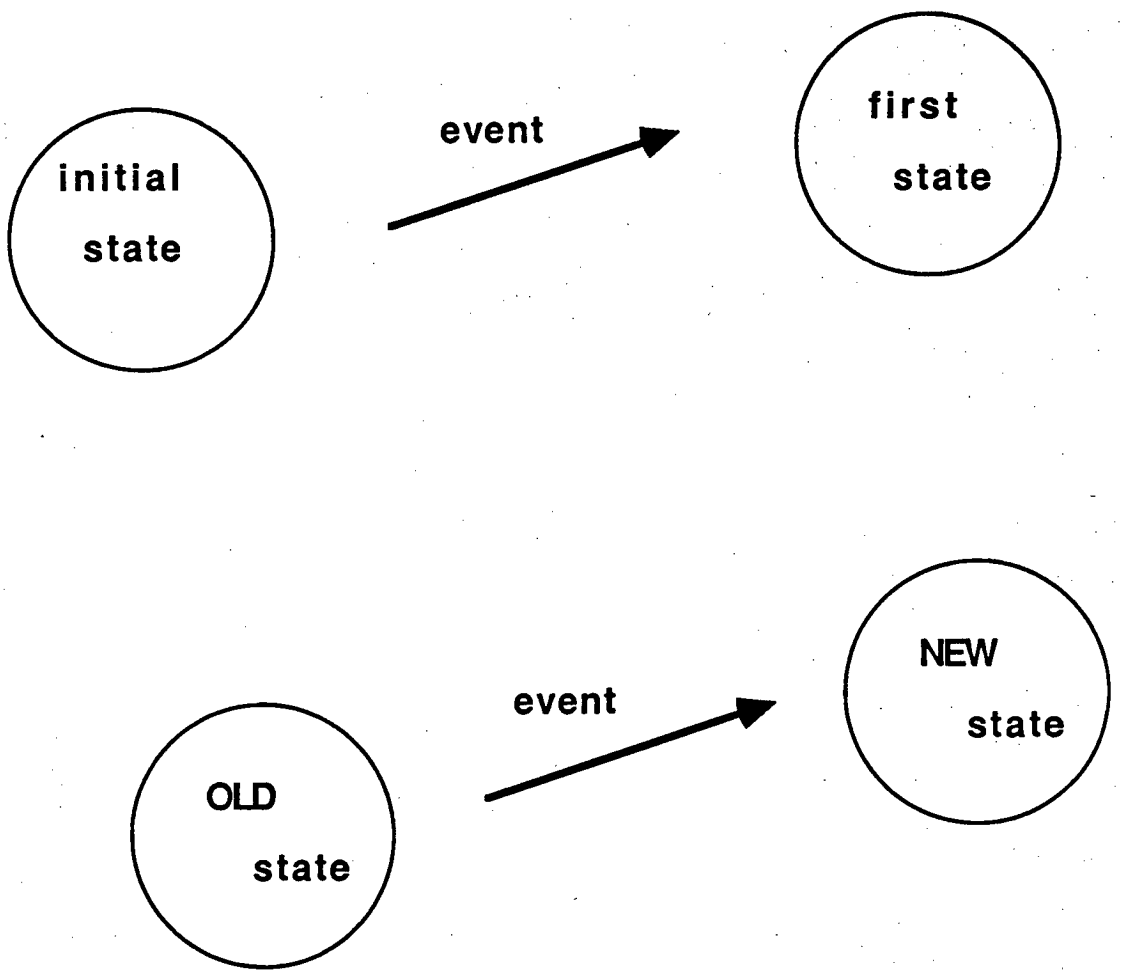
level (S) <= level (C) write C

State Transformations (Rules)

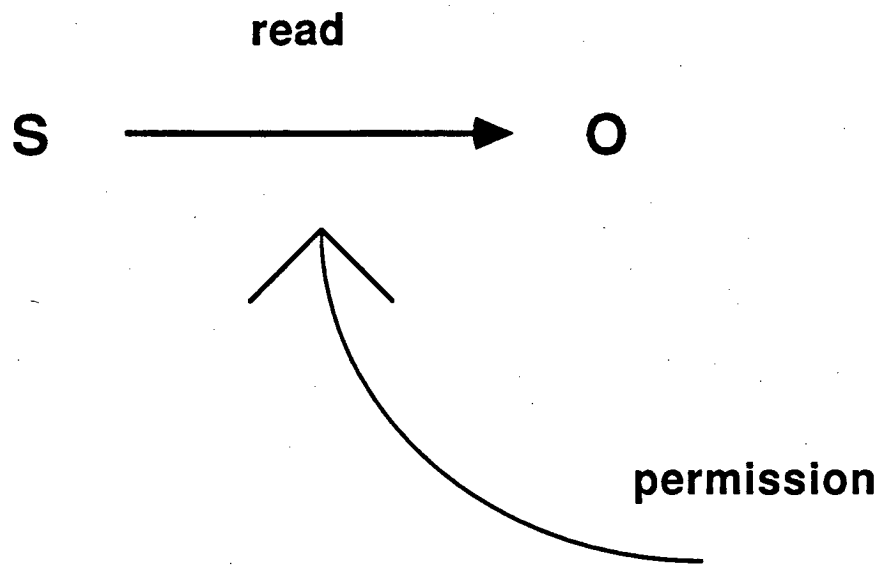
Get_Read	Release_Access	Create_Object
Get_Write	Give_Access	Delete_Object
Get_Execute	Rescind_Access	
Get_Append	Change_Current_Level	
	Change_Object_Level	

Underlying State Machine Model

State = (accesses, permissions, levels, subjects, objects)



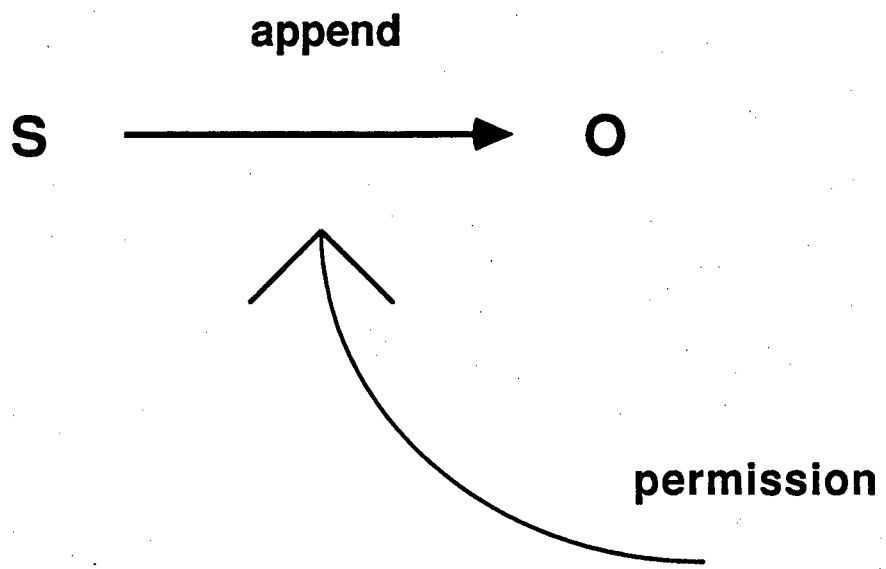
RULE 1 Get_Read



1. MaxLevel (S) >= Level (O)
2. S trusted OR Current Level (S) >= Level (O)

Result : add access to Access Matrix

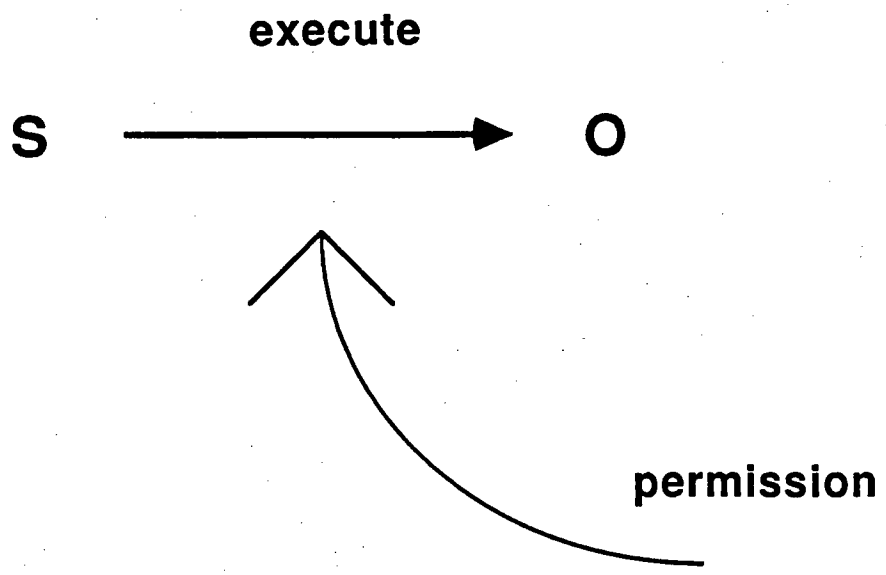
RULE 2 Get_Append



1. S trusted OR Level (O) >= Current Level (S)

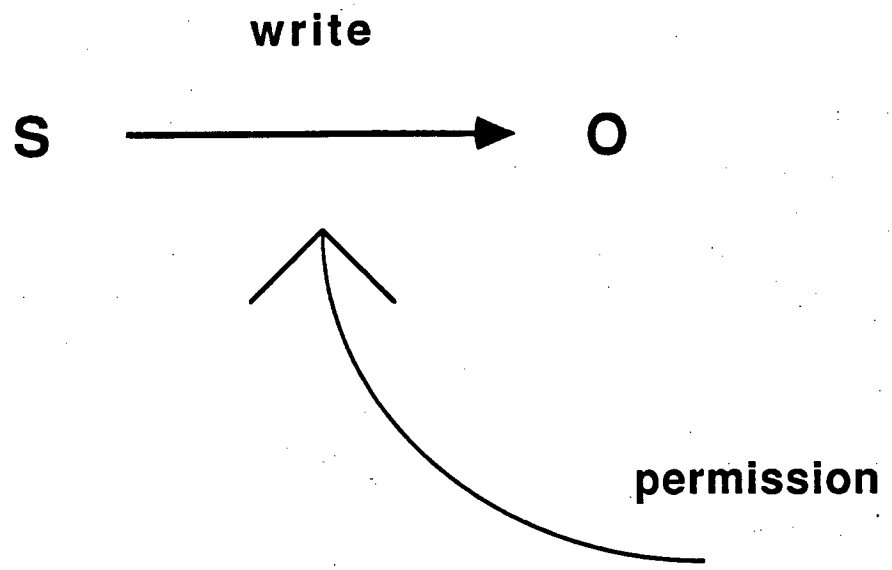
Result : add access to Access Matrix

RULE 3 Get_Execute



Result : add access to Access Matrix

RULE 4 Get_Write



1. S trusted and MaxLevel (S) >= Level (O)

2. S not trusted and Level (S) = Level (O)

Result : add access to Access Matrix

RULE 5 Release_Access

release A

Result : delete entry in Access Matrix

Release_Access

Release_Access(S, O, A) means that S releases access to O in mode A.

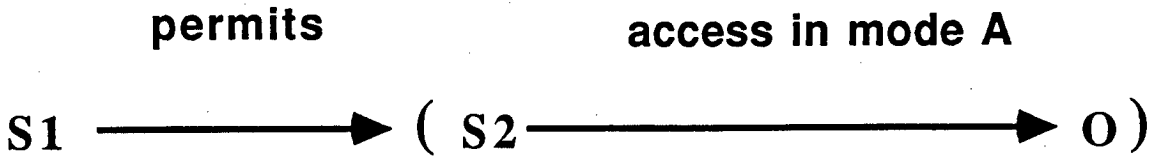
Here is the VERUS specification of this rule.

```
VAR x_access : Access;
CONST Cx_access : Access;
CONST Cx_access2 : Access;
CONST Celt : Elt;

DEFINE ReleaseAccess( subj, obj, x_access ) : BOOLEAN
BY
  AND
  {
    FOR elt
      IF AND
      {
        SubjectOf(elt) = subj;
        ObjectOf(elt) = obj;
        AttributeOf(elt) = x_access;
        InEltSet( elt, CurrentAccess(OLD) );
      }
      THEN NOT InEltSet( elt, CurrentAccess(NEW))
      ELSE
        IF NOT InEltSet( elt, CurrentAccess(OLD) )
        THEN NOT InEltSet( elt, CurrentAccess(NEW))
        ELSE InEltSet( elt, CurrentAccess(NEW));


    $ We also need to specify that no other state functions change:

    SamePermissions;
    SameCurrentLevel;
    SameObjectLevel;
    SameParent;
    SameHierarchy;
  };
```

Here is the VERUS proof that Release_Access preserves security.

```
s Declare instantiation constants.

CONST Csubj, Csubj1, Csubj2, Csubj3 : Subject;
CONST Cobj, Cobj1 : Object;

PROVE
  IF AND
  {
    StateRequirement(OLD);
    EXIST subj EXIST obj EXIST x_access
         ReleaseAccess( subj, obj, x_access );

  }
  THEN StateRequirement( NEW );
{
  PROVE SimpleSecurity( NEW );
  {
    EXIST subj EXIST obj EXIST x_access
         ReleaseAccess( subj, obj, x_access );

   SimpleSecurity( OLD );
  };

  PROVE StarProperty( NEW );
  {
    EXIST subj EXIST obj EXIST x_access
         ReleaseAccess( subj, obj, x_access );
    StarProperty( OLD );
  };

  PROVE DiscretionarySecurity( NEW );
  {
    EXIST subj EXIST obj EXIST x_access
         ReleaseAccess( subj, obj, x_access );
    DiscretionarySecurity( OLD );
  };
};
```

RULE 6 Give_Permission



Comments : Defined differently in different places

GIVE (S1, O) not completely defined--means that S1 can give permissions regarding O.

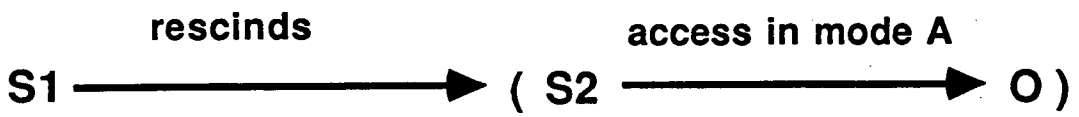
1. O is not ROOT and S1 has write access to Parent (O)

OR

2. O is ROOT and GIVE (S1, O)

Result : add entry to Permissions Matrix

RULE 7 Rescind_Permission



RESCIND (S1, O) is not completely defined--means that S1 can rescind permissions regarding O.

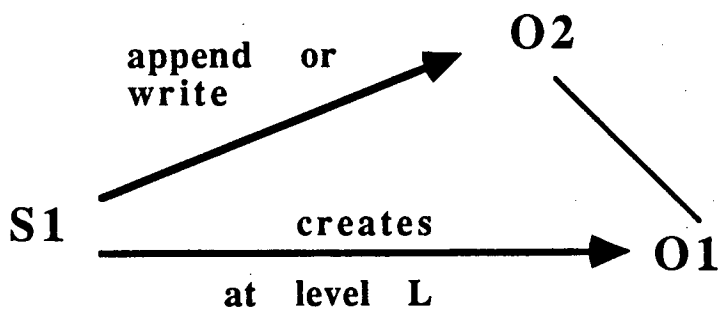
1. O is not ROOT and S1 has write access to Parent (O).

OR

2. O is ROOT and RESCIND (S1, O)

Result : delete entry to Permissions Matrix and delete all applicable entries in Current Access Matrix

RULE 8 Create_Object



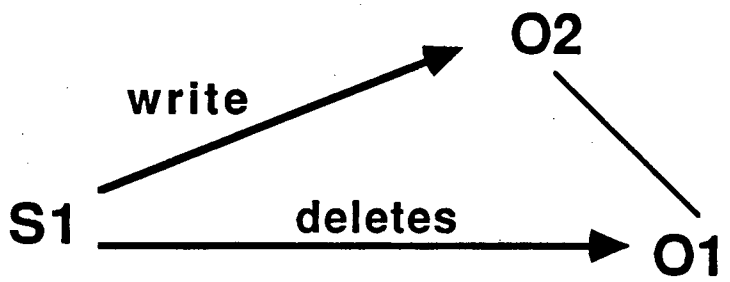
1. S1 has write or append access to Parent (O2).

2. L >= level (O2)

Comment : Evidently ROOT must exist in the initial state.

Result : a new object

RULE 9 Delete_Object



1. O1 is not ROOT and S1 has write access to Parent (O1)

Comment : Evidently, can not delete ROOT

Result : Delete O1 and subtree under O1.
Delete all applicable accesses from
the Access Matrix and all applicable
Permissions from the Permission Matrix.

Rule 10

Change_Subject_Current_Level

Current level (S) is changed to L.

1. MaxLevel (S) >= L
2. S trusted OR all current accesses by S obey the star - property assuming the new current level of S

Result : Current level of S is L.

Rule 11

Change_Object_Level

S ⟶ (Level (O) = L)

1. S trusted AND current level (S) >= level (O)

   OR

   current level (S) >= L >= level (O)

2. If any subject T has access to O in read or write mode, then MaxLevel (T) >= L

3. For all current accesses to O, the star - property still holds assuming the level of O is L

4. The compatibility property still holds assuming that the level of O is L

5. CHANGE (S, O) is true

Comment : CHANGE (S, O) is not formally defined in the model, but it means that S is permitted to change the level of O. No motivation is given for the asymmetry in 1.

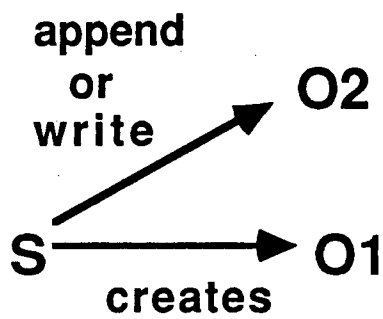
Result : The level of O is L

A Problem With Hierarchies

Scenario : S creates O (S not trusted)

S requests write access to O

Create_Object

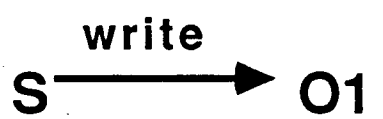


level (O1) >= level (O2)

star-property implies level (O2) >= level (S)

thus level (O1) >= level (O2) >= current level (S)

Get_Write



level (S) = level (O1)

All this implieslevel (S) = level (O2) = level (O1)

B - Level Secure Unix File System

- Processes (privileged, child)
- **B1 Security**

  Security Policy

  Descretionary Control (modes)

  Mandatory Control (labels)

  simple security

  star-property

  Accountability (logon)

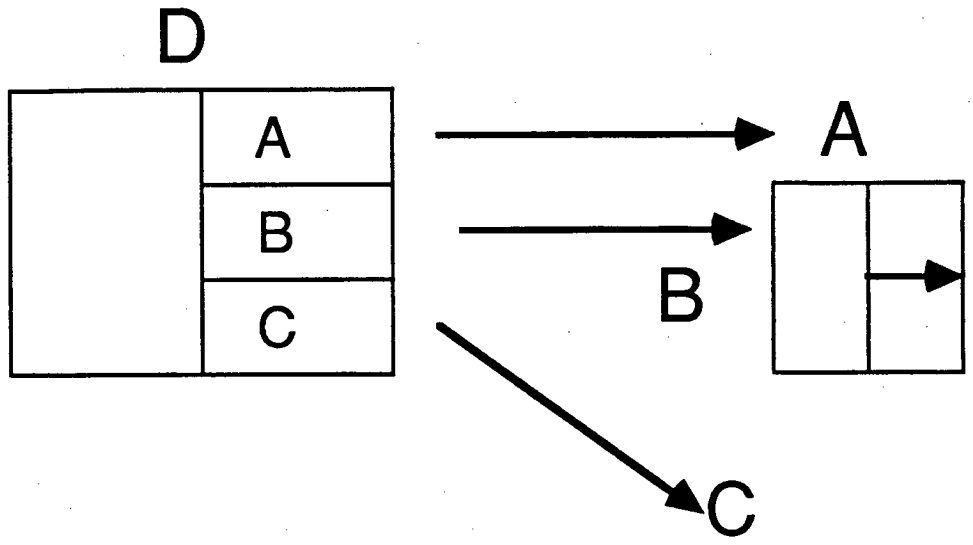
  Assurance (domain protection, model, testing)

  Documentation

Discretionary Access Control

-rwxr-xr--

Directories Are Not Containers



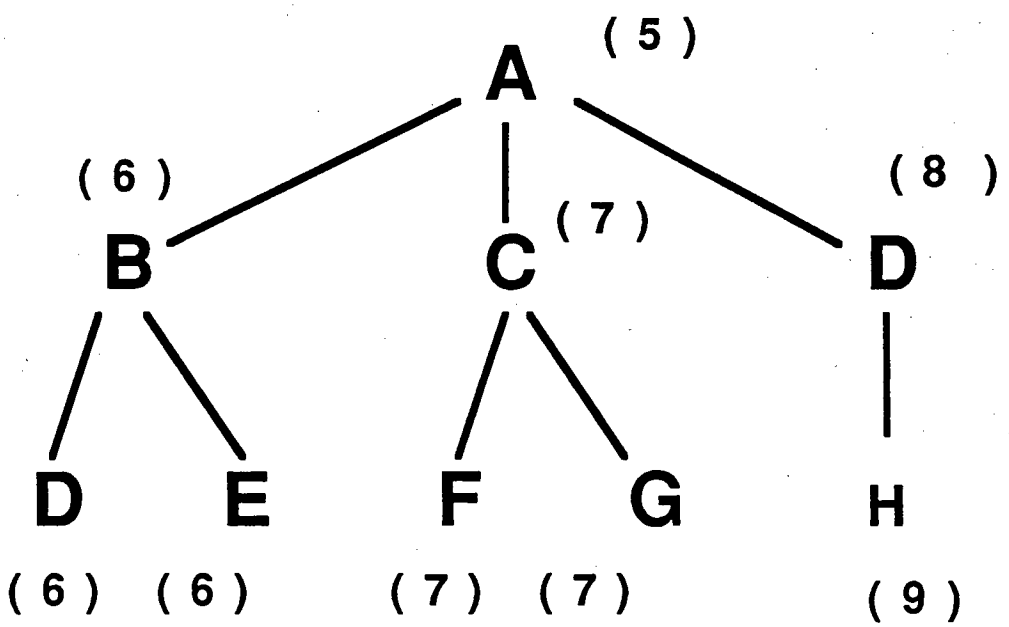
Directories Are Corridors :

execute: (search, if you can)

read: (list, if you can)

write: (create/delete, if you can)

Single - Level Directories



A Single - Level Directory has all files at the same level as the directory and all directories are single level at the same level.

No processes can create a file in a single-level directory at a level different from that of the dirctory.

Mandatory Policy For Directories (Rules About Labels)

In the rules that follow, U denotes any user. D denotes a directory. F denotes either a directory or an ordinary file. L denotes a label (level), and L(U), say, denotes, the current label (level) of U.

In order for any access to be possible to a file, the user must have execute access to all directories superior to the file.

execute	If U gains execute access to D, then L(U) >= L(D).
read	If U gains read access to D, then U must already have execute access to D. If the name of a file F is displayed during a read access, then L(U) >= L(F).
write	If U gains write access to D, then U must already have execute access to D. If D is a single-level directory, then L(U) = L(D). If U creates or deletes a file F, then L(U) = L(F). If U deletes a file F, then U must already have discretionary access to F in write mode.

Mandatory Policy For Data Files (Rules About Labels)

In the rules that follow, U denotes any user. F denotes an ordinary file. L denotes a label (level), and L(U), say, denotes the current label (level) of U.

In order for any access to be possible to a file, the user must have execute access to all directories superior to the file.

execute	If U has execute access to F then L(U) >= L(F).
read	If U has read access to F then L(U) >= L(F).
write	If U has write access to F then L(U) = L(F).

Extra Mandatory Checks

- Root - Owned Files
- Protected Device Files