1961

1981

CONGRESSO ANNUALE
ANNUAL CONFERENCE

A.I.C.A.

ASSOCIAZIONE ITALIANA PER IL CALCOLO AUTOMATICO

PAVIA 23-25 Settembre 1981

atti

volume II

IMPLEMENTING MULTIPLE AUTHORIZATIONS BY EXTENDED CAPABILITIES

P. Corsini*, G. Frosini**, L. Lopriore***

* Istituto di Elettronica e Telecomunicazioni, Università di Pisa, Pisa, Italy
** Istituto di Informatica, Università di Ancona, Ancona, Italy
***Istituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy

In a protected system, we deal with a multiple authorization when a subject is allowed to access a protected object only if several other subjects agree. In this paper, such a problem is examined, and a solution is given with reference to an extended capability environment. The solution presented holds for every number of agreeing subjects. Application problems are briefly illustrated.

1. INTRODUCTION

In a computer system provided with a capability based protection mechanism, the entities to which accesses must be validated are referred to as objects, and the entities that access objects are referred to as subjects /1,2,3/. The ability of a subject to access a given object is demonstrated by the possession of a capability, that consists of a unique identifier for an object and a set of access rights on that object. A subject can authorize another subject to access a given object by passing a capability for that object to it.

In this paper, we will refer to memory segments as objects, and we will analyze the specific problem of multiple authorizations, which consists in making a subject able to access an object only if some other subjects agree in authorizing it /4/. The problem, that cannot be simply solved by a capability passing, will be approached by utilizing a generalized form of classical capabilities, called extended capabilities /5/.

2. CAPABILITY BASED ADDRESSING

The characteristics of capability environments have been widely described in literature. We will refer to a particular capability based addressing, as described in /6/. For the sake of clearness, let us briefly recall the most important features of such environment.

The objects to be protected are the segments of a segmented virtual memory SVM, and an address in a segment consists of a unique segment identifier ID and an offset W in the segment. A pair (AR,ID), where AR is a set of access rights on the segment identified by ID, is called capability. Possible access rights are READ and WRITE for segments containing data; EXECUTE for segments containing codes; TAKE, GRANT and ENTER for segments containing capabilities. The meaning of the access rights READ, WRITE and EXECUTE are self-explaining. The TAKE and GRANT access rights allow the loading and storing of capabilities from and to capability segments, respectively. The meaning of the ENTER access right will be clarified later. A set of capabilities forms a domain. A domain is generally structured as a rooted tree of capability segments, the root of the tree being called Base Capability Segment (BCS), and the other nodes Auxiliary Capability Segments (ACS's). In a capability segment relative to a node N there are capabilities for the capability segments relative to the sons of N, with the access rights TAKE and GRANT. At a given time, a process operates in a specific domain, and the pair {process, domain} is referred to as a subject. A subject enters another subject when the pertinent process switches from one domain to another.

A possible implementation of the referred capability based addressing requires that each processor is provided with n capability registers CR_0, CR_1, ..., CR_{n-1}, besides the classical resources. Moreover, the Program Counter PC must contain the index of a capability register and an offset. Some special instructions, that are now briefly described, must be provided:

- Loadcap CR_i, W, CR_j

It loads into CR_j the capability stored in the segment pointed by the capability contained in CR_i (at offset W). The capability stored in CR_i must have the access right TAKE.

- Transfer CR_i, W', CR_j, W"

It transfers the capability stored in the segment pointed by the capability contained in CR_i (at offset W') into the segment pointed by the

This work has been sponsored by the Convention of Selenia S.p.A. and Consiglio Nazionale delle Ricerche, Pisa, Italy.

capability contained in CR_j (at offset W"). The capabilities contained in CR_i and CR_j must contain the access rights TAKE and GRANT, respectively.

- Enter CR_i, W

It pushes the contents of the capability registers and of the Program Counter into a proper process stack. Moreover, it loads a capability for the segment pointed by CR_i, with the access right TAKE, into CR_1, and loads the capability stored in this segment (at offset W) into CR_0. Finally, it loads the Program Counter with the index 0 and the offset 0, and clears all the capability registers, excluding CR_0 and CR_1. The capability initially contained in CR_i must have the access right ENTER.

- Reenter

It restores the contents of the capability registers and the Program Counter with the quantities popped from the process stack.
A classical instruction has the form

- Do η to CR_i, W

It performs the action η on the segment pointed by the capability stored in CR_i (at offset W). The capability stored in CR_i must have an access right consistent with the action η.

Note that when a subject is entered, since CR_0 contains a capability for its Base Capability Segment, the first instruction to be executed is a Loadcap instruction with CR_i=CR_1. Note also that, due to the rooted tree structure of a domain, capabilities stored in Auxiliary Capability Segments of any level can be loaded into the capability registers, by executing successive Loadcap instructions.

In the next Section, we will modify the previous capability environment, by: i) introducing the concept of extended capability; ii) slightly modifying the previously described instructions; iii) introducing the new special instructions Newpseudo and Amplify and the new access right AMPLIFY.

3. THE CONCEPT OF EXTENDED CAPABILITY

The classical concept of capability can be generalized by introducing the concept of extended capability, as follows. An extended capability is a triplet (F_1,F_2,ID), where: i) F_1 (1 bit) specifies if the extended capability is a true-capability or a psuedo-capability; ii) F_2 represents a set of access rights for a true-capability, or an offset for a pseudo-capability; and iii) ID is a unique identifier for a memory segment. True-capabilities have exactly the same aim of classical capabilities. Instead, a pseudo-capability is utilized in order to point to a specific entry (i.e., the entry specified by the offset F_2) of the segment identified by ID. A process possessing a given pseudo-capability for an entry of a segment is only allowed to access that entry and only under further constraints that will be stated later in detail. Processes are prevented from forging true-capabilities: if they need a new segment, they can ask a special protection monitor for a new true-capability. Instead, they can be authorized to forge pseudo-capabilities.

All the previously introduced instructions must be modified to treat extended capabilities. In particular, all instructions, excluding the Loadcap and Transfer instructions, must abort if they work on pseudo-capabilities.

In order to understand the aim of pseudo-capabilities, let us introduce the new special instructions Newpseudo and Amplify, and the new access right AMPLIFY.

The Newpseudo instruction generates a pseudo-capability for an entry of a capability segment CS_1, and loads it into another capability segment CS_2. The execution of such an instruction terminates correctly only if the subject possesses a true-capability for CS_1, with the access right TAKE, and a true-capability for CS_2, with the access right GRANT. More precisely, the instruction has the form

- Newpseudo CR_i, W', CR_j, W"

Its execution phase involves the following actions (Fig. 1):

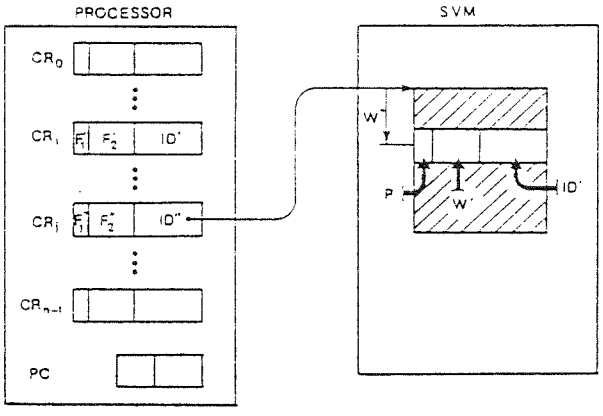

Fig. 1 Actions involved in the execution of the Newpseudo instruction

1) consider the extended capability stored in CR_i, say (F'_1,F'_2,ID'), and the extended capability

stored in CR_j, say $(F"_1,F"_2,ID")$;

2) if (F'_1,F'_2,ID') is a true-capability, if F'_2 contains the access right TAKE, if $(F"_1,F"_2,ID")$ is a true-capability and if $F"_2$ contains the access right GRANT, then the execution phase continues, otherwise an access violation occurs;

3) construct a pseudo-capability $(P,F*_2,ID*)$, where $F*_2$=W' and ID*=ID', and store it into the location specified by the pair (ID",W").

The Amplify instruction requires, as its operands, a true-capability for a capability segment with the access right AMPLIFY, and a pseudo-capability for an entry of the same segment. The result of the instruction execution is the loading into a capability register of the extended capability pointed by the pseudo-capability. More precisely, the instruction has the form

- Amplify CR_i, CR_j

Its execution phase involves the following actions (Fig. 2):

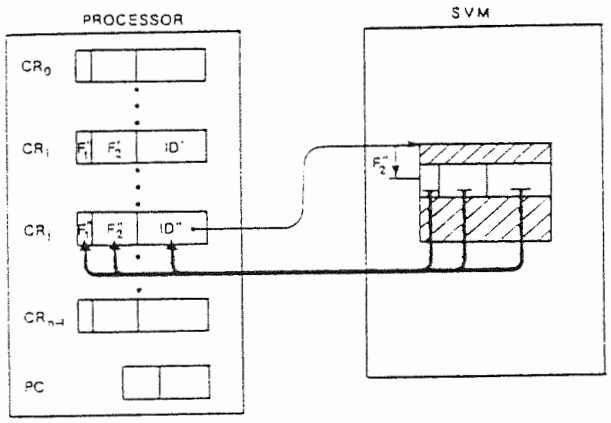

Fig. 2 Actions involved in the execution of the Amplify instruction

1) consider the extended capability stored in CR_i, say (F'_1,F'_2,ID'), and the extended capability stored in CR_j, say $(F"_1,F"_2,ID")$;

2) if (F'_1,F'_2,ID') is a true-capability, if F'_2 contains the access right AMPLIFY, if $(F"_1,F"_2,ID")$ is a pseudo-capability, and if ID"=ID', then the execution phase continues, otherwise an access violation occurs;

3) by utilizing the pair $(ID",F"_2)$, find the addressed extended capability in the pertinent capability segment, and load it into the capability register CR_j.

The main application of extended capabilities and the Amplify instruction is the efficient implementation of objects of abstract type /5/. In the next Sections, a further application is given, that is the solution of the multiple authorization problem.

4. MULTIPLE AUTHORIZATIONS

The problem of multiple authorizations consists in making a subject able to access a given segment R of SVM in a manner η only if several other subjects are all consentient. More precisely, a subject s* must be allowed to obtain a true-capability for R, with access rights consistent with the action η, under the constraints that each one of q agreeing subjects $s_0,s_1,\ldots,s_{q-1}$ i) does not possess the suitable true-capability for R in its domain, and so it cannot autonomously authorize s* to access R in the stated manner; and ii) performs a proper action, that shows its agreement for the considered access.

Let us now solve the stated problem in the case q=2 (Fig. 3). Each of the two agreeing subjects

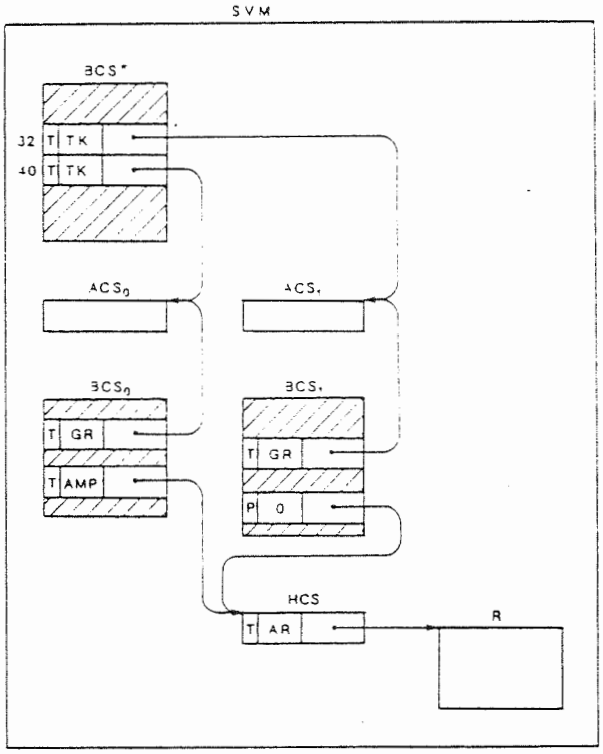

Fig. 3 Virtual memory situation for implementing multiple authorizations: the case q=2

s_i, i=0,1, possesses in its Base Capability Segment BCS_i a true-capability for a capability segment ACS_i with the access right GRANT. Moreover, the subject s* possesses a true-capability for ACS_i with the access right TAKE.

The segment ACS_1 will represent an unidirectional communication channel for the transmission of extended capabilities from s_1 to s*. A further capability segment must be provided, called Hidden Capability Segment HCS, constituted of a unique entry. This entry contains a true-capability for R, with access rights consistent with the action η. Note that BCS_0 and BCS_1 do not contain any true-capability for the segment R. Moreover, they cannot take such a true-capability from HCS, as they do not possess the required true-capability for this segment. Instead, BCS_0 contains a true-capability for HCS with the access right AMPLIFY, while BCS_1 contains a pseudo-capability for the unique entry of HCS.

The actions to be performed by s_0 and s_1 to give their agreement to the access of s* to R are the following:

- s_0 must transfer the true-capability for HCS with the access right AMPLIFY into ACS_0;
- s_1 must transfer the pseudo capability for the unique entry of HCS into ACS_1.

The subject s* can load the true-capability for R, contained in HCS, into a capability register, by executing the following instructions (we refer to the memory situation shown in Fig. 3):

```
Loadcap    CR1,32,CR2
Loadcap    CR1, 0,CR2
Loadcap    CR1,40,CR3
Loadcap    CR3, 0,CR3
Amplify    CR2,CR3
```

The first two instructions load the true-capability for HCS with the access right AMPLIFY into CR_2. The third and fourth instructions load the pseudo-capability for the unique entry of HCS into CR_3. The Amplify instruction loads the true-capability stored in HCS into CR_3.

The above solution can be generalized to the case of q agreeing subjects, with $q > 2$. In Fig. 4 the case q=4 is shown. A communication channel must be provided from each agreeing subject to s*: let ACS_i, i=0,1,...,q-1 be such communication channels. Moreover, (q-1) Hidden Capability Segments HCS_r, r=1,2,...,q-1 must be provided, which are linked to form an unidirectional list: the last segment of the list HCS_{q-1} contains a true-capability for R with access rights consistent with the action η, while each of the other segments contains a true-capability with the access right AMPLIFY for the next segment in the list. Each of the q agreeing subjects contains in its Base Capability Segment an extended capability for a Hidden Capability Segment. More precisely, i) BCS_0 contains a true-capability for HCS_1, with the access right AMPLIFY; and ii) $BCS_1, BCS_2, \ldots, BCS_{q-1}$ contain a pseudo-capability for the unique entry of $HCS_1, HCS_2, \ldots, HCS_{q-1}$, respectively. The actions to be performed by the q agreeing subjects to allow s* to access the segment R are the following:

- s_0 must transfer the true-capability for HCS_1 with the access right AMPLIFY into ACS_0;
- $s_1, s_2, \ldots, s_{q-1}$ must transfer the pseudo-capability for the unique entry of $HCS_1, HCS_2, \ldots, HCS_{q-1}$ into $ACS_1, ACS_2, \ldots, ACS_{q-1}$, respectively.

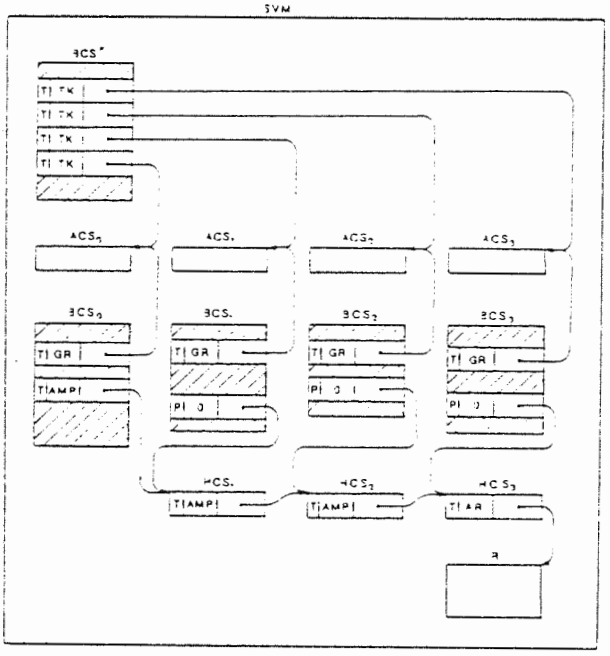

Fig. 4 Virtual memory situation for implementing multiple authorizations: the case q=4

When s* has obtained the pertinent extended capabilities from the agreeing subjects, it can gain the true-capability for R, stored in HCS_{q-1}. More precisely, this true-capability can be obtained by performing successive steps:

- in the first step, s* utilizes the extended capabilities stored in ACS_0 and ACS_1 for

obtaining the true-capability stored in HCS_1, by means of an Amplify instruction;
- in the r-th step, r=2,3,...,q-2, s* utilizes the true-capability obtained in the (r-1)-th step and the pseudo-capability stored in ACS_r for obtaining the true-capability stored in HCS_r, by means of an Amplify instruction;
- in the (q-1)-th step, the true-capability for R, stored in HCS_{q-1}, can be so obtained.

5. CONCLUDING REMARKS

The multiple authorization technique just described can be typically utilized to implement protected forms of rendez-vous mechanisms. In fact, subject s*, waiting for an access authorization on a given resource, can only proceed after all the consentient subjects $s_0, s_1, \ldots, s_{q-1}$ have reached the point in their evolution where they perform their agreeing actions. The mechanism is protected, in the sense that it is not possible for s* to proceed without the needed agreement even if it wants to do so, because it does not possess the required capabilities. This is in contrast with the traditional synchronization mechanism based only on semaphores, that can be simply overcome by malicious agents. Semaphores are also to be utilized in the presented solution, and they allow the agreeing subjects to notify to s* that the agreement has occured.

Note finally that, once s* has received the pertinent access authorization, the authorization itself cannot be revoked unless an explicite mechanism for access right revocation is provided by the protection system /7,8/. In this case, the class of problems in which the multiple authorization technique is useful enlarges substiantially, to include all the cyclical situations in which a given authorization must hold only during one cycle. An interesting application is represented by protected forms of synchronization.

REFERENCES

/1/ M.A. Harrison, W.L. Ruzzo, "Protection in Operating Systems", Comm. ACM, Aug. 1976.

/2/ T.A. Linden, "Operating System Structures to Support Security and Reliable Software", Computing Surveys, Dec. 1976, pp. 409-445.

/3/ G.S. Graham, P.J. Denning, "Protection-Principles and Practice", Proc. AFIPS 1972, SJCC, pp. 417-429.

/4/ J.H. Saltzer, M.D. Schroeder, "The Protection of Information in Computer Systems", Proc. IEEE, Sept. 1975, pp. 1278-1308.

/5/ P. Corsini, G. Frosini, L. Lopriore, "Implementation of Abstract Objects in a Capability Based Addressing Architecture", Technical Report R-81001/P/E, Convenzione Selenia-CNR, Feb. 1981.

/6/ P. Corsini, G. Frosini, F. Grandoni, L. Lopriore, "Capability Based Addressing: An Overview", Proc. AICA Annual Conference '80, AICA, Oct. 1980, pp. 164-176.

/7/ V.D. Gligor, "Review and Revocation of Access Privileges Distributed Through Capabilities", IEEE Trans. Software Engineering, Nov. 1979, pp. 575-586.

/8/ P. Corsini, G. Frosini, L. Lopriore, "Distributing and Revoking Access Authorizations on Abstract Objects: a Capability Approach", Technical Report R-81003/P/E, Convenzione Selenia-CNR, Mar. 1981.