J. Parallel Distrib. Comput. 115 (2018) 29–40

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Password-based protection of clustered segments in distributed memory systems

Lanfranco Lopriore

Dipartimento di Ingegneria dell'Informazione, Università di Pisa, via G. Caruso 16, 56126 Pisa, Italy

HIGHLIGHTS

- We refer to a distributed system consisting of nodes connected by a local area network.
- We consider the distribution, verification, review and revocation of access permissions.
- A form of protected pointer, the handle, is used to reference clusters of memory segments allocated in the same node.
- A handle referencing a given cluster includes a password and a selector of the segments in that cluster.
- We take advantage of a parametric one-way function for password generation.

ARTICLE INFO

Article history:
Received 7 April 2016
Received in revised form 12 January 2018
Accepted 15 January 2018
Available online 2 February 2018

Keywords:
Access right
Distributed system
Parametric one-way function
Protection
Revocation
Segment

ABSTRACT

With reference to a distributed system consisting of nodes connected by a local area network, we consider the problems related to the distribution, verification, review and revocation of access permissions. We propose the organization of a protection system that takes advantage of a form of protected pointer, the handle, to reference clusters of segments allocated in the same node. A handle is expressed in terms of a selector and a password. The selector specifies the segments, the password specifies an access right, read or write. Two primary passwords are associated with each cluster, corresponding to an access permission for all the segments in that cluster. A handle weakening algorithm takes advantage of a parametric one-way function to generate secondary passwords corresponding to less segments. A small set of protection primitives makes it possible to allocate and delete segments in active clusters, and to use handles to access remote segments both to read and to write. The resulting protection environment is evaluated from a number of viewpoints, which include handle forging, review and revocation, the memory costs for handle storage, the execution times for handle validation and the network traffic generated by the execution of the protection primitives. An indication of the flexibility of the handle concept is given by applying handles to the solution of a variety of protection problems.

© 2018 Elsevier Inc. All rights reserved.

1. Introduction

We refer to a distributed architecture consisting of nodes connected by a local area network. The network topology is inessential. We hypothesize that, in each node, the primary memory is partitioned into two regions, *private* and *shared*. The private memory can only be accessed by software components being executed in that node. The shared memory, which is reserved to interprocess communication, can also be accessed by software components running in the other nodes, albeit in a strictly controlled fashion. We shall not consider the mechanisms for the control of private memory accesses; instead, we shall concentrate on shared memory protection, with special reference to the problems inherent in the distribution, verification, review and revocation of access permissions. These are major problems in the design of any protection system. Our solution complies with a segmented view of the shared memory.

A *segment* is a contiguous memory area completely defined by a *base* and a *length*. The base is the address of the first storage unit reserved for the segment, the length expresses the segment size. Segments are the elementary unit of information transmission and sharing between the nodes. Two operations are possible on a segment, to read the segment contents and to replace these contents. Protection applies to *local* segments allocated in the shared memory of same node as the software component attempting the access, as well as to *remote* segments allocated in the shared memory of the other nodes.

In a classical protection paradigm, active entities, called *subjects*, generate access attempts to protected, passive entities, called

E-mail address: lanfranco.lopriore@unipi.it.

https://doi.org/10.1016/j.jpdc.2018.01.003
0743-7315/© 2018 Elsevier Inc. All rights reserved.

objects [18,24,35]. The system associates a set of *access rights* with each object. A subject aimed at accessing a given object to execute one of the operations defined for that object must possess an access right permitting successful accomplishment of that operation; if this is not the case, the access attempt generates a protection violation, and fails. A *protection domain* is a set of access rights for correlated objects. A subject being executed in a given protection domain can access the objects, taking advantage of the access rights included in that domain.

We shall hypothesize that a subject can be a scheduled computation (a process), or, in an event driven environment, a software routine activated by a hardware interrupt [22]. Segments are the objects on which protection is exercised. Two access rights are defined for segments, *read* and *write*. A subject that holds access right *read* for a given segment is allowed copy the segment contents from the shared memory of the node where the segment is allocated into the private memory of the node where that subject is running. Similarly, a subject that holds access right *write* for a given segment can overwrite the segment contents with quantities taken from its own private memory.

We consider segments grouped in clusters. A *cluster* is a collection of correlated segments, all contained in the shared memory of the same network node. A subject that holds access right *read* for a given cluster can access the segments of that cluster to read their contents; this is similar to access right *write* for segment write accesses.

1.1. Capabilities

A major problem in the design of a protection system is how to represent the access rights held by each subject. A classical solution is based on the concept of a *capability* [15]. This is a pair (G, AR), where G is an object identifier, and AR is a set of access rights. A subject that holds capability (G, AR) can access object G to carry out the actions permitted by the access rights in AR.

Several aspects of a practical implementation of the capability concept deserve in-depth consideration for their impact on performance and usability. These include capability segregation, weakening, review and revocation, and the memory requirements for capability storage.

1.1.1. Segregation

Subjects must be prevented from modifying capabilities, for instance, to add access rights to an existing capability, or even to change the object identifier to forge a capability for a different object. Several solutions to this capability segregation problem have been proposed [5,16,32]. In a segmented memory system, special segments, which we shall call *capability segments*, can be reserved for capability storage (in contrast, the *data segments* contain ordinary information items) [6,12]. A *capability list* is a collection of capabilities for correlated objects; a capability segment contains a capability list. The instruction set of the processor will be enlarged by the addition of special *capability instructions* for capability processing. Capability segments can only be accessed by using the capability instructions; if an ordinary data instruction is used, an exception of violated protection is raised.

In an alternative approach, a 1-bit *tag* is associated with each memory cell, which specifies whether this cell contains a capability or an ordinary information item [1,11,31]. A cell tagged to contain a capability can only be accessed by using the capability instructions. This approach requires memory banks specialized to contain the cell tags, and is contrary to the requisite of hardware standardization [20].

1.1.2. Weakening

A subject that holds a capability for a given object can transfer a copy of that capability to another subject. In this way, the recipient acquires the whole access privilege specified by that capability. In fact, a capability copy is indistinguishable from the original, and possession of the copy is equivalent to possession of the original. On the other hand, it may well be the case that a subject wishes to transfer only part of the access rights included in the original capability. In a classical capability environment, this means that the instruction set of the processor should include a capability instruction to modify the access right field in a strictly controlled fashion, excluding access right amplification. In a common approach, the access right field features one bit for each access right; if asserted, the given bit denotes the presence of the corresponding access right. The access right weakening instruction will permit to clear (but not to set) these bits.

1.1.3. Review and revocation

A subject that receives an access privilege in the form of a capability is in the position to transmit it further. It follows that capabilities tend to disperse throughout the system, and it is hard to keep track of all existing copies of the original capability. In a distributed system, this problem is exacerbated by the possibility that copies spread to different nodes. A relevant problem is access right revocation: a subject that created a given object should be in the position to withdraw the access privileges distributed for this object [2,8,9]. An essential property is that the effects of a revocation should propagate to all the subjects that hold the access privilege being revoked (*transitive* revocation). In a distributed system, this means that revocation should extend across node boundaries. Other desirable properties are the abilities to limit revocation to a specific subset of the access rights (*partial* revocation), to revoke different access privileges for the same memory area independently of each other (*independent* revocation), and to restore the original privileges through the same mechanism as for revocation (*temporal* revocation).

Several solutions to the access right revocation problem have been devised [18]. Examples are a propagation graph associated with each capability, which keeps track of all successive transferrals of this capability between subjects [8]; temporary capabilities with short lifetimes, which must be renewed periodically to avoid implicit revocation [14]; and a centralized reference monitor associated with each object, which keeps track of the subjects that hold access permission for this object [28]. These solutions tend to impair a basic advantage of capability protection, i.e. simplicity in access right transmission between subjects.

1.1.4. Memory requirements

A subject that is granted access privileges for a number of distinct objects has to hold a capability for each of these objects. The resulting memory requirements tend to be high in percentage. This is especially the case if the system should support a large number of small-sized objects [7,33], if we are aimed at exercising protection at a high level of granularity. Consider, for instance, a capability list that grants access permissions to a group of segments. We have to reserve a capability segment to contain the capability list. This capability segment is a memory waste, and a significant complication in access privilege management.

1.2. Password capabilities

Password capabilities are a remarkable improvement on the capability concept [2,3,10,17]. A password capability is a pair (G, P), where G is an object identifier, and P is a password. A set of passwords is associated with each protected object, and each password corresponds to an access privilege expressed in terms of a set of access rights. If password P matches one of the passwords associated with object G, the password capability grants the access privilege corresponding to the matching password.

Download English Version:

https://daneshyari.com/en/article/6875042

Download Persian Version:

https://daneshyari.com/article/6875042

Daneshyari.com