M^2S: A Machine for Multilevel Security

Bruno d'AUSBOURG, Jean-Henri LLAREUS

CERT/ONERA
Département d'Études et de Recherches en Informatique
2,avenue E.Belin
B.P. 4025
31055 Toulouse Cedex France
email : {ausbourg,llareus}@tls-cs.cert.fr

Abstract. In this paper we describe the architecture of a computer machine ensuring a protection for data and processes of various classification levels, concurrently running on behalf of various cleared users. The security, enforced by a hardware security subsystem, is based on an internal information flow control that prevents building any illicit channel. Mechanisms and services of standard operating systems may be built on this machine. It permits also to build and manage multilevel data structures and multilevel computations which are able to satisfy the highest security requirements of new applications.

1 Introduction

It exists relatively numerous products and systems which are reputed to enforce various security functions, and to rate levels defined in the Orange Book [5]. But they did not succeed to convince potential users. Today, it clearly appears that only a high security degree, based on strict control of access and a full control of the internal information flows, can be able to allow computations for sensitive information.

In this paper, we describe the general architecture of a machine ensuring a protection for data and processes of various classification levels concurrently running on behalf of variously cleared users. This protection is exerted for both confidentiality and integrity. The choices made for the architecture are founded on a precise and formal definition of security considered through the use of security levels. The security properties which outcome from this theory exert some constraints on levels and are enforced by a security subsystem: its good functioning is sufficient to ensure the whole security. The independence between the subsystem and the computing unit in the machine permits to offer mechanisms and services of a standard operating system to the user, ensuring an effective portability for existing applications.

A first section shows how the control of causal dependencies, by use of level attributes, permits to ensure the whole system security. The second section exhibits the main principles which outcome from this approach and have directly an influence upon the definition and implementation of a security subsystem. The architecture satisfying these principles is described in the third section. The fourth section exhibits some principles in order to structure a such operating system taking account of this architecture. The last section illustrates their application to a Unix system development through the description of implementation mechanisms for multi-level file systems and process systems.

2 Related works

The earlier works on the first secure systems, as described in [11] and [10], contributed to forge and then to implement the idea of a security kernel. In this approach, the security is mainly based on the first theories of Bell and La Padula [3] and Biba [2]. It stays entirely in the operating system area and this entails a great operating role for the security kernel: it performs basic system operations on which it enforces the security properties. The security kernel acts as a real operating subsystem in charge of preserving the whole security and constitutes a system layer with its own services and interface. The operating system, developed on this kernel basis, is in fact an other new layer with its own services adapted to the security kernel interface and providing the user with a rarely standard interface. To reach this goal, an other emulation layer becomes necessary, leading to a complex system, with a high cost and poor performances [9].

The idea of making a distinction between the security, relying on hardware mechanisms, and the operating system, appears with the SCOMP machine [7], the first A1 rated machine. The security still relies upon the Bell and La Padula definition and the security subsystem masters internal information flows by enforcing some controls over the management and the access to the memory segment descriptors stored in the MMU. This realization lets some covert channels opened but attempts to reduce their capacity by introducing noisy. This leads to decrease system performances. In addition, the developers of this machine have not considered appropriate the use of a standard operating system as Unix on a multi-level machine, so the given interface did not ensured a satisfying portability of programs.

More recently, the Lock machine appeared [4] founded on the non-interference model [8]. It is the most closest to M^2S. The mains differences lie in the security definition, in the kind of protection which follows from it and in the structure of the resulting operating services.

3 Protection by levels

3.1 Control of causal dependencies

The aimed security is able to protect both confidentiality and integrity of data and processes over the system. The formal definition given in [6] and [1] establishes, with regard to confidentiality for example, that a system is secure if and only if the set of all the objects that may be observed in the system by a subject s, $O(s)$, is included in the set of objects he has the right to observe $R(s)$:

$$O(s) \subseteq R(s) \qquad (1)$$

It is important to define $O(s)$ very closely. If not, a subject s can observe some object o not in $O(s)$. This object o can be used by a trap or a Trojan horse to disclose any secret information.

In fact, a user is able to perceive values of various objects inside the system. Some of them may have a finer granularity than files. For example, a user can observe the status value of processes, of data structures as a lock or a semaphore, of memory cells or of registers inside a disk controller. He can also observe duration of operations as a disk access, a memory access. Then he can observe the value of data at various given times, and perceive dates of their changes. Therefore, what may be really observed in the system is more than single objects, but *points (object,time)*. Indeed, saying that a point *(o,t)* may be observed by a subject *s* involves two kinds of possible observations which entail two kinds of communication channels:

- the *value* of the object *o* at time *t* may be perceived, and a storage channel is involved here;
- the *time* or date *t* at which the object *o* takes a given value may be perceived, and a timing channel is involved here.

So, *O(s)* comprises points *(o,t)* that must be understood as values of objects *o* at a given time *t*. Output objects may be directly observed by a user and then their associated points *(o,t)* belong to *O(s)*. These points are produced by computations from other points reflecting the state of internal objects. These internal objects are themselves produced by computations from input data. So, *O(s)* contains more than points that can be directly observed: it contains also the points on which the points that may be directly observed depend on. A precedence order on instants *t* defines as *causal* these dependencies in the model.

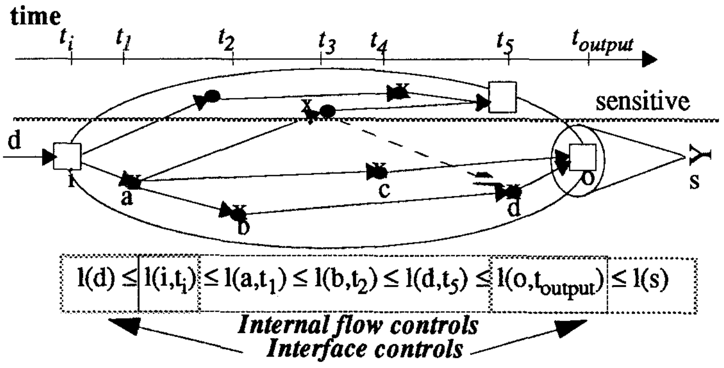


Fig. 1 Causal dependencies inside a system

For instance, Fig. 1 describes a system with an output object *o*, whose value is observed by a subject at time t_{output}. The value of this object *o* depends on previous values of objects reflected by points (d,t_5), (c,t_4), (b,t_2), (a,t_1), and finally (i,t_i) where *i* is an input object and t_i satisfies

$$t_{output} > t_5 > t_4 > t_2 > t_1 > t_i \qquad (2)$$

If the subject knows the internal system functioning, by observing *o*, he can deduce values of intermediate points on which (o,t_{output}) is depending causally: (d,t_5), (c,t_4), (b,t_2), (a,t_1), for example. So, he can deduce values of some input points (i,t_i) in the system. Then, it is possible for him to discover some input values which he would not have the right to observe. Except if the system ensures that all these points contain no sensitive information and are not computed from sensitive ones. In other words, a subject perceives no sensitive information if the system ensures the condition expressed in (1).

In order to maintain the set of objects that a subject s can observe, $O(s)$, in its rights $R(s)$, it is necessary to control causal dependencies inside the system.

3.2 Protection by levels

The use of levels allows to ensure a good mastering of dependencies and of the associated information flows inside the system. A security level (both in confidentiality and integrity) is attributed to objects and subjects. A subject with a clearance level $l(s)$ is allowed to observe only system points (p,t) whose level $l(p,t)$ satisfies:

$$l(p,t) \leq l(s) \qquad (3)$$

This inequality (3) must remain true for all points which are observable by the subject inside the system. Taking back Fig. 1 , this condition leads to conclude that the inequality (4) must be satisfied inside the system:

$$l(d) \leq l(i,t_i) \leq l(a,t_1) \leq l(b,t_2) \leq l(d,t_5) \leq l(o,t_{output}) \leq l(s) \qquad (4)$$

In particular, it is forbidden for a point (d,t_5) to causally depend on a point (x,t_3) with $l(x,t_3) \geq l(d,t_5)$. Remember that this causal dependence could be:

- the value of object d at t_5 depends on (x,t_3);
- the value of time t_5 at which the object d takes a particular value depends on $(x,t3)$.

This would enforce a potential information flow from the sensitive point (x,t_3) down to a not sensitive one (d,t_5) and would be contrary to the definition of the security previously given. In fact, it follows that point (o,t_{output}) depends on no sensitive point in the system and its observation will reveal no sensitive information.

So, inequalities at system interfaces, as described in Fig. 1 can be enforced by classical techniques of interface protection. The control of causal dependencies (including its temporal aspects) allows to make sure productions and elementary transfers of information until system points directly observed by a user. All information channels are involved (storage and timing) and it exists no potential covert channel. Values of levels constitute a public (not classified) information.

4 Architecture principles

This definition of security and of information flow control (by causal dependencies controlling) has been interpreted and implemented in M^2S. During developing, some principles guided the choices made for the architecture.

4.1 SSS: Security SubSystem

The whole security relies only upon the good functioning of a subset of hardware and software components constituting the *Security SubSystem (SSS)*. This SSS manages data and achieves necessary operations in order to maintain security properties all over the system. This principle is similar to the TCB one expressed in [5]. In particular the SSS conforms to the reference monitor properties: it cannot be bypassed and its integrity is protected.

4.2 SSS in hardware layer

The flow control model can be interpreted in various system abstraction layers. However, at a given layer N, specifications of operations inside layer N may verify security properties issued from the model. But these N-operations rely on data and functions specified at layer N-1.

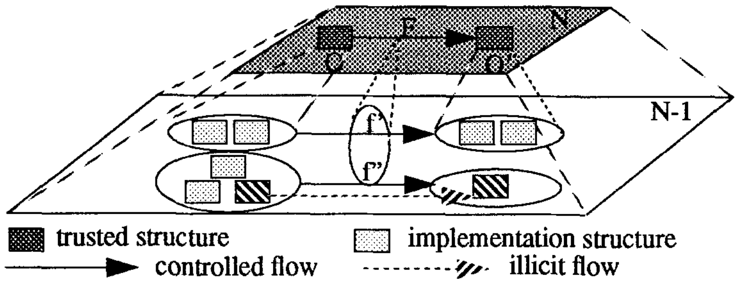


Fig. 2 Introduction of illicit flows in a refinement structure

The layer N-1 specifications refine layer N specifications and introduce additional operations and data structures that can be used to transfer information and build illicit information flows.

A way to avoid difficulties related to refinement structures and illustrated by Fig. 2 consists in directly staying inside the hardware layer. This is possible because semantics of security levels are sufficiently simple to be considered at this layer.

4.3 Level assignment

The used programming model is a classical one. It combines a processor P with an address space A. P addresses A when it executes elementary transfers to external devices (memory, registers of a device controller...).

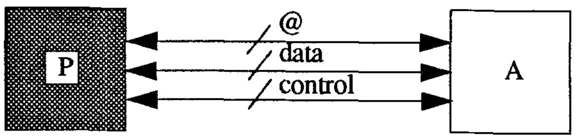


Fig. 3 Traditional programming model

The processor P, extended by buses, is an active entity and acts as a subject over the system. Objects that can be observed are composed by the processor registers and the cells of A. Objects values at various times are the points of the system. Levels are assigned to the processor and the cells of A. Levels are themselves security objects. The processor level determines the *current level cl* of the whole system. Levels divide A in various partitions. Each partition may be reached by the processor according to the *cl* value, the requested access mode and the rules of flow control.

4.4 Levels and control of elementary transfers

The state of the system is reflected by the status of processor registers, address, data and control buses. Elementary transfers into the address space, or interrupt signals travelling to the processor and carried up by the control bus constitute the internal flows of information.

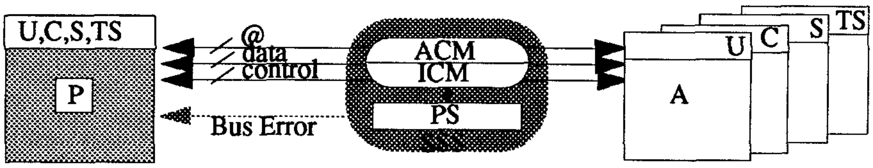


Fig. 4 Elementary transfer controls inside the system

So, controls executed by the SSS include two main modules. Each one is implemented by making use of specific hardware components under control of a Security Processor PS. This PS owns resources necessary to store and manage security data.

ACM: Access Control Module. This module inspects in real time the states of the address and control buses and determines which states are allowed according to the security data stored in SSS and according to the rules related to these transfers. The status of buses is examined during each elementary cycle. In case of an illicit one, the cycle may be interrupted by the PS which issues a Bus Error request destined to the processor P.

Briefly, at current level cl, a read (or write) cycle to an address of level n_a will be allowed only if following conditions (4) (or (5)) are satisfied:

$$cl \geq n_a \qquad (5)$$

$$cl \leq n_a \qquad (6)$$

This module comprises an additional specific component in charge of verifying some transfers whose addressing mode is more complex and uses transfers on the data bus: for instance, access to disk data blocks.

ICM: Interrupt Control Module. This module acts as a filter for the interrupt signals emitted by peripheral devices located in the address space. If the sender is an object at level l_o, the interrupt signal is assigned a level $l_i = l_o$. It is transmitted to the processor P when the current level cl satisfies:

$$cl \geq l_i \qquad (7)$$

The interrupt signal is suspended until condition (7) becomes valid. In fact, in order to handle this signal more easily, a stronger condition is waited for:

$$cl = l_i \qquad (8)$$

4.5 Management of level objects

Levels of the processor or of cells in A constitute objects in the system. Therefore, a level is assigned to them. In a simplification way, this level is the minimal level, l_{min}. So, the level of an object is a public information. Its modification is constrained by the flow control rules. In particular, these rules ensure that the value of an object *(o,t)* can depend only on informations contained in objects *(o',t')* such as

$$l_{(o,t)} \geq l_{(o',t')} \qquad (9)$$

When *(o,t)* is a level object, the property expressed by (9) requires (10) to be satisfied:

$$l_{min} \geq l_{(o',t')} \qquad (10)$$

This means that the value of a level, at a given time t, must only depend on public information. This is possible only if, at a public level, a strategy is enforced to reserve resources in advance for classified levels.

Modification of levels for cells in address space. Partitioning the address space is achieved at public level. Therefore it is possible to declare a space partition at a given level l_p for a time t eventually infinite. This declaration enables the SSS to manage the corresponding level data and to ensure its coming back to the l_{min} value, after t.

Such a backward is equivalent to a downgrading of the partition. So it is a constrained operation and the SSS is in charge of it. Flow control rules require the partition be cleared during this operation. So, the value of this partition at level l_{min}, after downgrading, depends only on informations of level l_{min}.

Current level modification. Modifying the value of the current level consists in determining, at public level, different values for *(cl,t)* objects in the future. In other words, it's necessary to plan a temporal multiplexing for current level. The processor will adopt a functioning according to the Fig. 5 .

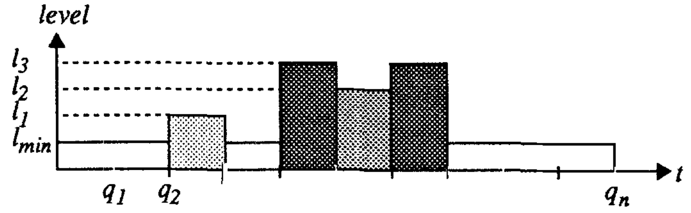


Fig. 5 Current level temporal multiplexing

The SSS allocates temporal quanta q_i required for various levels. When, at a given time t, known at level l_{min}, the SSS modifies the current level cl, flow control constraints demand objects of processors (particularly registers) to be cleared when cl decreases. This forces the value of these object to depend only on information of level l_{min}.

5 M²S architecture

5.1 General architecture

The general architecture is defined by Fig. 6 . It is founded by insert of the SSS, driven by the Security Processor PS, a MC68010, in a bus cutting. This PS can be reached by the processing unit, a MC68020 processor, through the micro-machine and the available coprocessing interface.

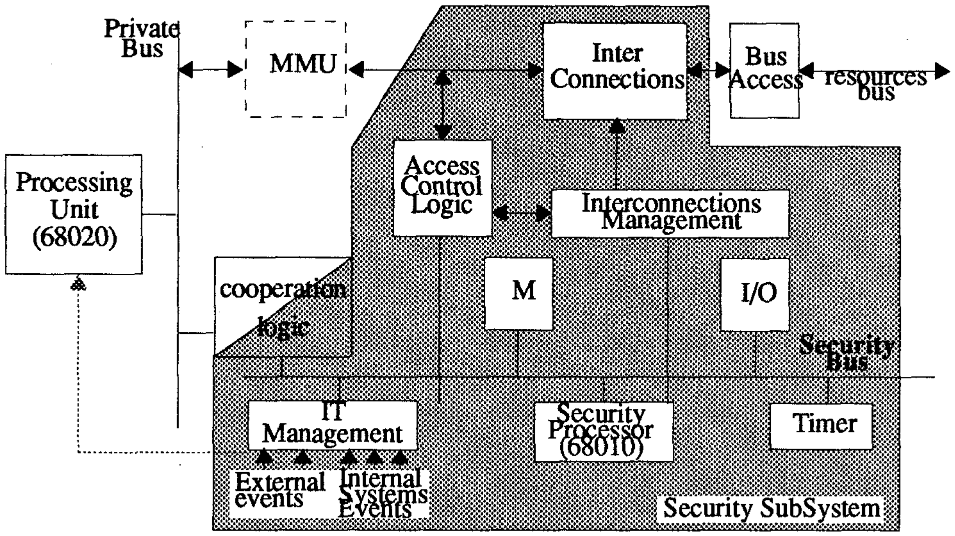


Fig. 6 General architecture for the machine

The SSS, under the control of a security processor, enforces the flow control rules and manages data on which this control relies. It cannot be bypassed because it intercepts all the accesses executed by the processor to its resources. Moreover, data and security programs are located in a security space S separated from the processing unit area P. P can reach S only through the achievement of a coprocessing dialogue, fixed by micro-machine and mastered by the Security Processor PS. This ensures the integrity of the SSS.

Three functional blocks are composing the SSS. They are discussed in the following paragraphs.

5.2 Cooperation between SSS and processing unit

The processing unit can reach the SSS space only by executing a coprocessing dialogue with the Security Processor PS. A cooperation logic, founded on a double access memory, permits data exchanges between both processing and security spaces.

The processing unit is able to transmit request blocks to the SSS on order:

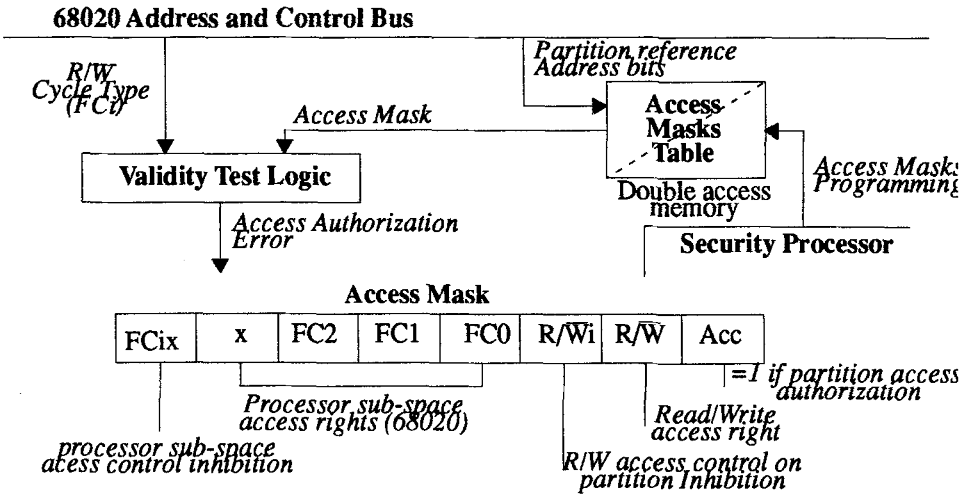
- to retrieve security data;
- to make reservations in advance for classified levels of resources.

In all cases, the Security Processor stays mastering security data management: the processing unit has no possibility to retrieve or to falsify security information without the SSS knowing.

5.3 Controls of elementary flows of information

The functioning of any machine makes use of two kinds of internal elementary flows. First involves flows initiated by the processing unit when it does accesses to its own address space. ACM module attends to control these flows. Second kind involves flows initiated by any hardware devices located in the address space when emitting interrupt signals to the processor. ICM module attends to control these internal flows.

Controls of elementary accesses. The controls of an elementary access relies on a single comparison between the real state of the bus and a mask of allowed configurations for it. These masks are located in a double access memory.



68020 Address and Control Bus
R/W
Cycle Type (FCi)
Access Mask
Partition reference Address bits
Access Masks Table
Double access memory
Access Masks Programming
Validity Test Logic
Access Authorization Error
Security Processor
Access Mask
FCix
x
FC2
FC1
FC0
R/Wi
R/W
Acc
=1 if partition access authorization
Processor sub-space access rights (68020)
Read/Write access right
processor sub-space access control inhibition
R/W access control on partition Inhibition

Fig. 7 Hardware mechanisms for flow controls

The address bits select an entry in a mask table according to the reached partition in the address space. A mask is a byte expressing the allowed access modes for the selected address. For instance:

- access forbidden;
- read and/or write access;
- allowed access in all processor sub-spaces;
- allowed access in only particular sub-spaces (FC_i).

The value returned by the mask is compared to the real one found on the control bus. The PS computes masks according to the security data it knows (here levels) and control rules it is in charge of enforcing (here, flow control rules).

The access control as described by Fig. 7 is exerted during each memory cycle. According to the result, the cycle may be carried on or interrupted. In this case, the elementary access control mechanisms interrupts the PS which in turn may interrupt the processing unit by initiating a Bus Error signal.

The independence between the flow control devices and the rules establishing their good programming permit to have at one's disposal a mechanism able to enforce many other security policies than the single multi-level security.

Interrupt signals controls. The second mechanism involves controlling the interrupt signals issued to the processing unit by cells in the address space. A single flip-flop battery, programmed by the PS, enables interrupt requests by filtering them according to their level l_i (equal to the emitting address space cell) and *cl*.

Filter lets the interrupt requests carrying on only when $l_i = cl$. In the opposite case, interrupt requests are retained until to be handled by the processing unit when allowed by *cl*.

5.4 Trusted Paths

It is necessary for the user to have at his disposal a trusted path to the SSS. This path may be used in order to exchange security data with the SSS. Mainly for:

- his identification and his authentication;
- his session level reservation.

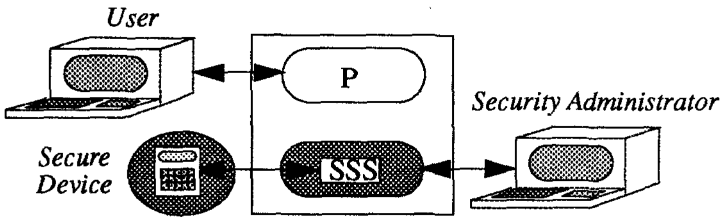


Fig. 8 Trusted path between SSS, user, and Security Administrator

This trusted path is implemented in a Secure Device (SD) which is able to display messages from the SSS and to read security data given by the user. Directly connected to the SSS, it ensures the integrity of these security data and offers an extension of the SSS until the user.

There is also a trusted Path directly between the SSS and the Security Administrator console which acts as an interface between them. The SA can exert security functions in order to enter or to modify security data.

6 Structuring tools for a multi-level operating system

6.1 Domains by use of levels

The architecture discussed in § 5 defines a machine provided with domains which can enforce a confinement for data and processes. The use of levels, based on simple semantic concepts, in order to define these domains, permits the implementation of the SSS mechanism inside the hardware layer. The result is a machine provided with multi-level domains.

Therefore, every program, and particularly the operating system, running on the processing unit, is constrained by hardware controls. So, the operating system is in charge of managing resources and providing the user with an access interface to a new virtual machine distinguished by its ability to:

- make partitions of resources by levels, including the processor resource;
- enforce a strict flow control between these levels, including temporal flows.

Taking account of these facts entails some principles that are convenient to apply when building an operating system this machine.

6.2 Multiplexing data structures according to levels

The multilevel functioning mode of the machine enables all the mechanisms in charge of controls to enforce rules expressed in § 3 . Access to any data is constrained by these rules. Multiplexing data structures by levels allows to take account of this fact during the system development stages.

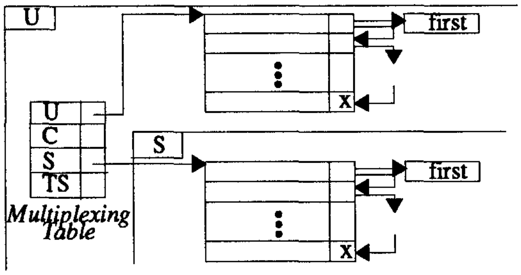


Fig. 9 Multiplexing by levels a process table

Fig. 9 illustrates how multiplexing a process table by levels. Data structure is split up in two components. One is located in a public memory part (Unclassified) and the other is located in a secret memory part (Secret). The structures of the two components are similar. Access is achieved through a multiplexing table indicating, for each active level, the base address of the corresponding level structure.

This multiplexing organisation is founded on the hypothesis that levels are minimal level objects, l_{min} (here U). Such level data are managed at public level U. So, the existence and the address of secret data structures may be known at public level.

At a given current level *cl*, only data structures at level *cl* are allowed to be managed. So, the only processes able to be scheduled at current level *cl* are *cl* processes.

Therefore processes are submitted to a double scheduling strategy. First, the SSS allocates various current levels to the processor resource. These levels are active over the system for lengths of time previously established at public level. Then, for each current level, the operating system allocates the processor resource to ready processes that are managed inside the *cl* process table.

6.3 Blindly writes

The flow control rules allow some exchanges of data between levels. In the context of a multilevel operating system, it may be necessary, in order to achieve synchronisation and communication, to have any information sent from a level l_{inf} to a level $l_{sup} \geq l_{inf}$ that must be not observed. Two cases may occur.

Write to a known destination address. That's a write operation into memory from a level l_{inf} to a level l_{sup}. It is allowed by flow controls and will not be interrupted by SSS. Typically, this procedure may answer a need to set flags in a data structure at level l_{sup}.

Complex operation to a data structure managed at l_{sup}. In this case, the previous mechanism is unsatisfactory. Inserting an information inside the receiving data structure implies a call to specific management functions related to this structure. This needs to access management data at level l_{sup}. SSS will forbid such an access.

However, a possible way in order to come back to the previous case consists in allowing level l_{sup} to retrieve an information at level l_{inf}.

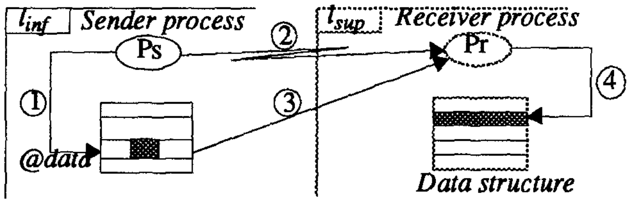


Fig. 10 Writing into a level l_{sup} data structure

Inserting a l_{inf} data into a l_{sup} data structure can be carried out in four steps.

- 1 Sender process P_s at level l_{inf} produces *data* to transmit and keeps it at $@_{data}$.
- 2 Sender process P_s signals receiver process P_r at level l_{sup} eventually providing him with $@_{data}$.
- 3 P_r retrieves data at level l_{inf}
- 4 P_r inserts *data* into the receiving data structure.

This hardly constrained mechanism permits to achieve the reservation of resources and the creation of objects at higher levels.

6.4 Anticipating hardware controls

The operating system is submitted to requirements of the hardware controls. In order to avoid to cause Bus Error signals related to security faults, it's necessary to take account of the hardware functioning inside system layers.

So, the functioning rules enforced by the SSS may have any influence upon the structure of traditional algorithms in operating systems, causing some semantic modifications to them, due to their integration into a multi level environment.

For instance, access to a file is generally carried out by the use of a descriptor *desc* kept in a memory descriptor table T_{desc}. When closing this file, a function *free_desc* allows freeing the entry allocated to *desc* in T_{desc} and updates the descriptor upon disk.

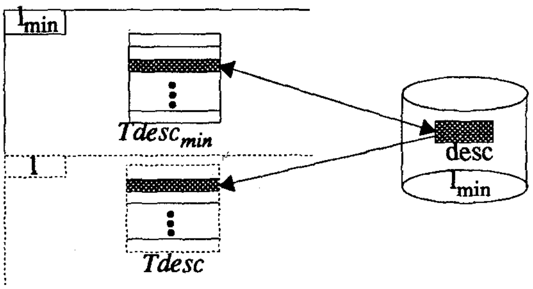


Fig. 11 Freeing a file descriptor in multi level mode

Assume the file is at level l_{min}, stored on disk at level l_{min}, the file descriptor *desc* inherits the file level l_{min}. Loading in memory and freeing the descriptor are achieved at the same level. Then, the operating functioning enforced by *free_desc* is a standard one.

Assume now that file reading is done at a given current level $cl > l_{min}$.Then descriptor *desc* will be read and loaded into descriptor table T_{desc} of level *cl*. The call to *free_desc* function will cause the T_{desc} entry freeing. But the algorithm must inhibit the update operation on disk because it would be blocked and interrupted by the SSS.

Such an algorithm must reflect the following algorithmic structure:

```
function free_desc (desc);
begin
if level(desc) == current-level () then disk_write(desc);
free_Tdesc(desc);
end;
```

6.5 Specific functions for multilevel management

Taking account of these mechanisms of partitioning by levels inside the operating system may be achieved by definition of specific functions and primitives. Principally around three main points.

Management of level data. A first set of functions must allow the operating system to acquire or, conversely, to request the modifications of security data, and particularly: level of users, current level, level of memory parts and level of peripheral devices.

Management of multilevel resources. Partitioning the address space by levels is enforced by the SSS. This also can be done through the use of an operating system service. So, it's advisable to build commands and functions allowing to achieve it. Particularly providing functions enabling to do multilevel configuration of memory and configuration of a multilevel file system.

Management of multilevel data and processes. Semantics of traditional operating system primitives do not allow to make level breaking inside data structures and so, do not allow to build multilevel data structures. Then it's necessary to forge functions and primitives in order to be able, at level l_{inf}, to create or to delete objects at level $l_{sup} \geq l_{inf}$. These operations concern the existence of objects and not their content. The existence (including its duration) is managed at level l_{inf} and depends only on information at level l_{inf}. So, exerting these operations causes no information flow from level l_{sup} to level l_{inf}.

7 Application examples to the Unix operating system

The operating system kernel developed for M^{2}S at CERT/ONERA uses Unix mechanisms, data structure, and operating functions. It offers to the user a set of interface primitives with additional primitives answering needs expressed at § 6.5 . Their building is submitted to principles discussed in § 6 . We illustrate the use of these techniques within the context of two operations requiring multilevel data structures. These operations are in the area of the management of multilevel file systems and the multilevel management of processes : creating a secret directory at public level and creating a secret process by a public process.

In order to simplify, creating a classified data or process structure is achieved at public level. So, the only level breaking that may be built is $(l_{min} \rightarrow l)$ with $l > l_{min}$. This is possible because the level of a level object is public. A feasible generalisation for these mechanisms would allow to build any kind of level breaking such as $(l_1 \rightarrow l_2)$ with $l_1 < l_2$.

7.1 Making a secret directory at a public current level

The Unix file system structure is based on a tree-like organisation of directories and regular files. Each one is reached through a descriptor inode that contains management and implementation data for the referenced structure. The table of loaded inodes in memory reflects the reachable file system. This table permits to retrieve data blocks implementing files and directories, directories keeping the links of the global tree structure.

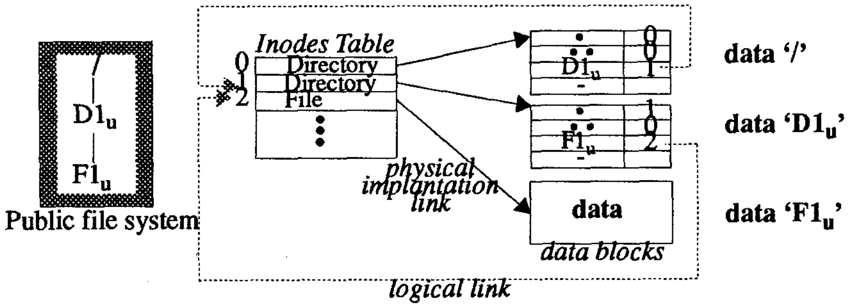


Fig. 12 Logical structure of a public file system

A structure as the one described in Fig. 12 can be extended in order to make a secret directory under the root, for example $D2_s$. Such an operation is based on the following techniques:

- *multiplexing* data structures between levels U and S.
- *reservation of resources* in advance and *blindly writes*;
- use of a specific primitive: *smkdir (directory,level)*.

Multiplexing by level the inode table splits it up into a secret memory area and a public one. The public multiplexing table provides with addresses of tables on each level. The secret area and data structures related to it are managed at a secret current level. At public current level, the logical structure linked to the public subset of file system is nearly similar to the one described in Fig. 12 .

The main difference resides in the *reservation* of the 10 first entries in the inode table. They are reserved in order to achieve making secret level directories. The ten first secret inode entries correspond to a reservation area for public inode entries. This secret area is intended to be use for directories or files created at public level.

At public level, achievement of primitive *smkdir("$D2_s$",secret)* causes the public inode table be searched for a free entry amongst the reserved ones for making secret directories. Inode 0 is found and initialized as a *reservation inode* for $D2_s$ directory. It contains management data related to this directory creation. Such a reservation inode permits to know, at public level, the secret directory existence and the whole information related to its creation. Then a blindly write up to the corresponding entry in the secret inode table initializes it at a value reflecting the creation at public level. The algorithm of the *smkdir* primitive stops here. At this stage, the only existence of the $D2_s$ directory is known by both levels public and secret.

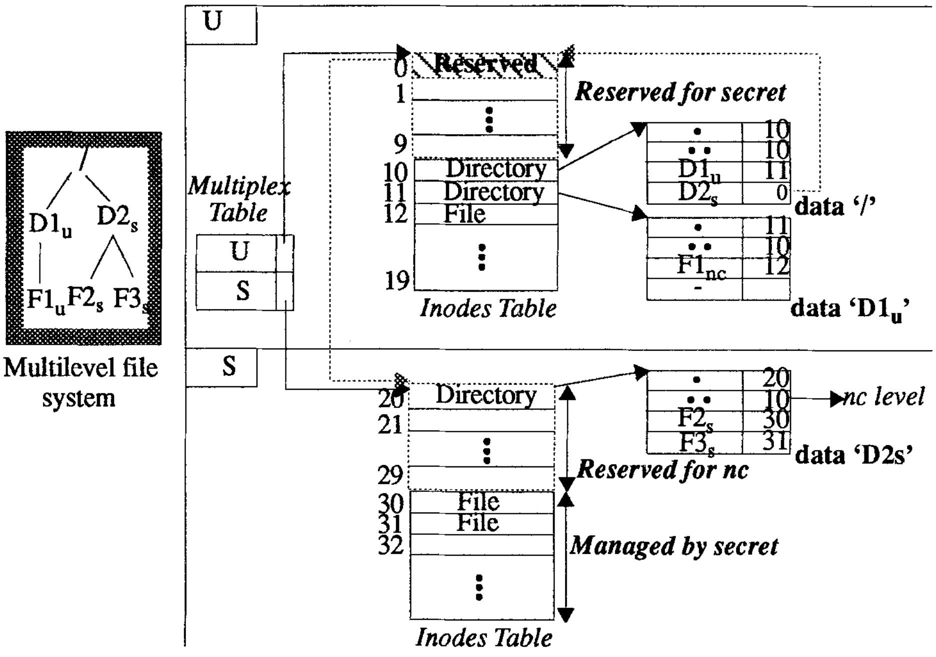


Fig. 13 Logical structure of a multi level file system

At secret level, the chosen option consists in achieving the secret data block allocation and the full inode initialization when opening the secret level directory. This operation calls the access function *namei*. This function searches the file tree structure for the inode corresponding to a logical pathname. Finding the corresponding reservation inode, it finds the inode as made from public level and allocates the secret data blocks. Then, a classical directory management at secret current level allows to create a secret subtree under secret root $D2_s$. In particular, files $F2_s$ and $F3_s$ are made and managed in a way in accordance with the one expressed by Fig. 12 .

Deleting $D2_s$ directory, as making it, modifies the public information of its existence. This operation may be achieved at the only public level. It consists in freeing the public reservation inode. The logical access path to the directory is then broken and the directory is lost for all levels.

From an operational point of view, the user will have previously deleted, at secret current level, the linked to $D2_s$ subtree. If not, secret data structures stay allocated though logically unattainable. They will be freed when opening a new directory on this same entry.

The achievement of such a level breaking in the file system remains a rare operation. Its more realistic use consists in building great partitions inside the file tree structure, each one used as a working area for each concerned level.

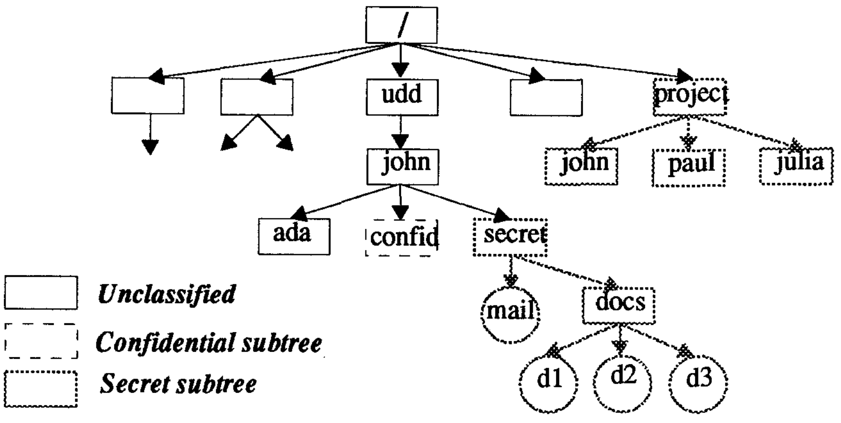


Fig. 14 Multi level file system

An organisation as illustrated by Fig. 14 is founded on an equally multilevel organisation for the disk file system. Briefly, it is based on the same multilevel structuration principles. So, it provides with the ability to make a single file system, ensuring a strict confinement for data amongst various levels. It provides also the ability to implement objects composed by variously classified data: for example, multi level files.

This single file system is a main difference with data structures provided by the Lock architecture [12]. In Lock/ix system, the multi level file system is based on the management of several file systems, each one managed and implemented at a single level. Moreover, this organisation is in a good accordance with non interference principle.

7.2 Creating a secret process at public current level

The use of the same techniques allows to introduce level breaks inside the tree structure of processes. Fig. 15 illustrates how creating a secret process at public level by the use of a forged primitive: *sfork(level,duration)*. This primitive accepts a *level* parameter and a length of time *duration* parameter that will be used by SSS in order to reserve timing quanta for requested level.

Data structure multiplexing is applied to the process table that is located on both levels, secret and public. Functions of memory reservation allow, at public level, to reserve secret memory blocks. Allocating and freeing them is achieved by means of the table planned for this use.

Creation at public level is based on the reservation of resources necessary to achieve the secret process. This ability is provided by the availability of reserved entries in process tables.

At public current level, during execution of the *sfork* primitive, a free entry in process table is searched amongst the reserved ones to create secret processes. If one exists, it is initialized with the whole data related to the process creation. A secret memory area is also allocated in secret memory reserved at public level. A blindly copy is exerted into it from memory space allocated to the creating process. Then a blindly write into the corresponding entry in secret process table permits to describe its context, addresses of its allocated memory space, and then to declare it in a "*created at public level*" status.

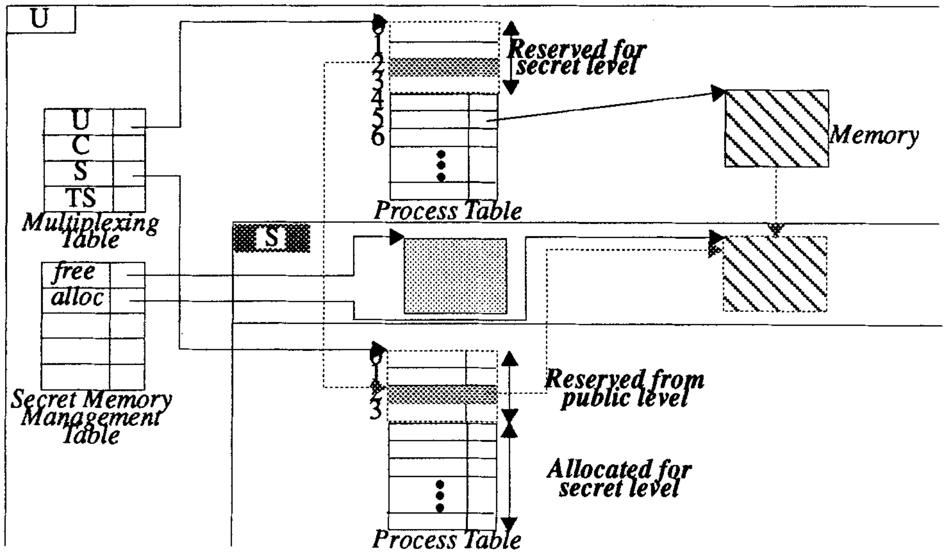

U
U
C
S
TS
Multiplexing Table
free
alloc
Secret Memory Management Table
S
0
1
2
3
4
5
6
Reserved for secret level
Memory
Process Table
0
1
2
3
Reserved from public level
Allocated for secret level
Process Table

Fig. 15 Creating a secret process at public level

So, when terminating the *sfork* execution, process existence and creation conditions are known at public level. This process is provided with a length of time for its life reserved and known at the same public level. Data needed for its management have been transmitted to secret level. At secret current level, the scheduling task is able to handle the newly initialised process table entry and to manage this process as a secret process. In particular, it will be able to insert it into the ready process queue.

This process may use the *exit* primitive to terminate. Semantics of this primitive are a little modified in order to take account flow control constraints. More precisely, when the terminating process has been created at public level, no ending signal is emitted to its father: this would be blocked by SSS.

That means a public process has at its disposal no information related to the activity of its secret child. He only knows its existence for a length of time declared in advance. This constraint about duration seems hard, but in fact, lengths of secret processing are generally well known.

Then, at public level, the process is declared terminated at the end of the length time initially declared for its life. Its public process table entry is made free as secret memory previously allocated to it.

This ability to create processes of various levels is also a difference with abilities provided by the Lock machine. In Lock/ix system, in accordance with non interference principles, a subject must declare its session level and all processes belong to this single processing level. There is no dynamic transfers of processes between levels.

8 Conclusion

The architecture of the machine M^2S detailed in this paper and developed at CERT/ONERA manages a unified structure for data and processes allowing them to coexist at different sensitivity levels. The model to which we refer in order to define the security is a causality model. The machine masters all causal dependencies (causality dependencies = (functional + temporal) dependencies) by controlling all elementary information flows. These last, enforced by the SSS, ensure there is not any covert channel (neither storage, neither timing) which can be enforced inside the system.

This paper discussed the protection offered from the point of view of confidentiality. The same level techniques are employed in the system to achieve an efficient protection from the point of view of integrity. In fact, the enforced controls seem rather hard for integrity. For instance, resource reservation requirements combined with duration controls are not necessary involved in integrity protecting. Nevertheless, these basic mechanisms ensure a confinement by level able to answer demanding needs for integrity.

The whole security is inside and only inside the SSS. There is no trusted part of the operating system needed for security reasons. The only constraint for it is to be adapted to the new virtual machine offered by the SSS functioning. In particular, it must take account of the separation and of the control of flows enforced by SSS in order to implement multilevel data structures and multilevel operating services as multilevel file systems, or multilevel files or multilevel process trees.

All standard mechanisms and functions of the Unix operating system (or any other OS) can be built on this architecture. So, the portability of existing applications is ensured. Moreover, mechanisms and functions are offered which permit to build other multilevel services or data structures. For example, it is possible to build, on a same file system, (and on a same disk partition) a multilevel file or directory tree, and also multilevel files (files with data of various levels). It is also possible for a public process to create a secret one. These dynamic breaks in levels are managed through the call of particular primitives defined and implemented in such an order.

This approach is hardware dependant. Multilevel security has sufficiently poor semantics to be interpreted in the hardware layer. This permits to obtain a very thin granularity in the flow controls that are enforced. And to avoid any illicit channel to be used, because all flows can be exhaustively controlled. The evolution watched in the processor architecture area incites to integrate, inside the same micro-machine, structures for data and codes interpretation ensuring the management of information necessary to enforce multi level security. This research goal would permit to obtain a new generation of processors able to achieve the confinement done on the existing machine.

In such an hypothesis, security can be perceived no more as a constraining task incumbent to the operating system, but as a structural feature the system has just to take account.

9 References

1. Bieber, F. Cuppens and G. Eizenberg : Fondements théoriques de la Sécurité Informatique. Rapport 2/3366.00/DERI, Centre d'Etudes et de Recherches de Toulouse, 1990.

2. K. J. Biba: Integrity Considerations for Secure Computer Systems, Technical Report ESD-TR-76-372, ESD/AFSC, Hanscom AFB, Bedford, Mass., 1977. Also MITRE MTR-3153.

3. D.E. Bell, L.J. LaPadula: "Secure Computer Systems : Unified Exposition and Multics Interpretation"- MTR-75-306, MITRE Corporation, Bedford, Mass, March 1975

4. J. M. Beckman, J.R. Leaman and O.S. Saydjari: LOCK trak : Navigating Uncharted Space, IEEE Symposium on Security and Privacy, Oakland, 1989.

5. Trusted Computer Systems Evaluation Criteria.Technical report DoD 5200.28-STD, National Computer Security Center, Fort Meade, MD, December 1985

6. G.Eizenberg: Mandatory policy: secure system model. In AFCET, editor, European Workshop on Computer Security, Paris,1989.

7. L. J. Fraim: Scomp, a solution to the Multilevel Security Problem. In IEEE Computer, July 1983.

8. J. Goguen and J. Meseguer: Unwiding and Inference Control. IEEE Symposium on Security and Privacy, Oakland, 1984.

9. Panel SessionKernel Performance Issues, Proc. Symp. Security and Privacy, IEEE Cat. No 81CH1629-5, Oakland, Calif.,1981.

10. E. J. McCauley and P. J. Drongowski: KSOS The Design of a Secure Operating System, AFIPS Conf. Proc., Vol 48, AFIPS Press, Montvale, N.J., 1979.

11. G. J. Popek: UCLA Secure Unix, AFIPS Conf. Proc., Vol. 48, 1979 NCC, AFIPS Press, Montvale, N.J., 1979.

12. M. Schaffer and G. Walsh: LOCK/ix : On implementing Unix on the LOCK TCB, 11th NCSC Conference, 1988.