CAPABILITY CONCEPT MECHANISMS AND STRUCTURE IN SYSTEM 250

D.M. ENGLAND
Plessey Telecommunications Research,
Maidenhead, G.B.

SUMMARY

The idea of Capabilities was introduced in 1966 in a paper by J.B.Dennis and E.C. Van Horn (Ref.2). Capabilities are described in a monograph on Time-Sharing Systems (Ref.3) by Professor M.V. Wilkes, whose work at Cambridge on Capabilities and Capability machines has been of substantial benefit to the Plessey Company's own developments. A number of Capability machines have been designed; of particular note is the work of R.S. Fabry (Refs. 4 and 5), who conceived the idea of copying Capabilities.

The System 250 processing system, developed by the Plessey Company Limited, is the first Capability machine to be realised in hardware. This paper discusses the concept of Capabilities as implemented in System 250. It explains the use of Capabilities in the construction of protected data structures. It describes some types of data structure that can be realised and how they are used in the construction of systems. The paper commences with a brief description of System 250 architecture.

This paper also forms an introduction to a companion paper (Ref.1), which describes the theoretical implications of Capabilities for the design of operating systems and the sharing and protection of information.

INTRODUCTION

Aims and Scope

1. System 250 is a multi-processor system. It has been developed by the Plessey Company for real-time applications, especially in the field of telecommunications. Perhaps the most important concept in the design of System 250 is the "Capability". The aims of this paper are as follows:-

(1) To describe the Capability concept and how it is implemented in System 250.

(2) To provide a basis for a companion paper (Ref.1) which discusses the application of Capabilities to the sharing and protection of information.

2. Although the scope of this paper is limited to the Capability concept, it is useful to begin with a brief description of System 250.

Outline of System 250 architecture

3. System 250 (Fig.1) is based on a multi-processor concept, using separate CPU and Store Modules. It employs a 24 bit word. An interconnection system permits any CPU to access any store word. The hardware configuration consists of a number of CPUs, each of which is connected to a number of Store Modules by means of a parallel data highway which is called a CPU Bus. The CPUs share the system workload by reference to a common work-list located in store.

4. High speed peripheral devices are interfaced to the CPU Buses in the same way as Store Modules. This means that they can be addressed like Store Modules. Each device looks to a CPU like a Store Module containing a small number of words, which are in fact the command, status, data, and address registers of the device controller.

5. Low activity devices, including teletypes, VDUs, and other kinds of terminal equipment, are interfaced to the system by means of a Serial Data Collection and Distribution Medium, composed of a network of data switches connected by serial transmission paths. External events are recorded by the Serial Medium and input by a CPU at some suitable. Consequently, there are no event interrupts in System 250.

6. No specialised input/output channelling modules are needed in System 250. Instead, the system is designed such that each CPU is capable of performing the channelling function by software. This permits the CPUs to share input/output channelling as part of the normal system workload. On completing a transfer to or from a device, a CPU is immediately capable of commencing another, and consequently there are no device interrupts in System 250.

7. The only types of interrupt which occur in System 250 are internal to a CPU and include fault, interval timer, and trap (see para.18) interrupts.

8. From the point of view of this paper, the storage system is of particular importance. It is composed of fast stores and disks and is organised as a Virtual Memory, which means that:-

(1) Storage space on disk is allocated dynamically by the Operating System in segments of arbitrary size.

(2) Transfers between fast store and disk occur automatically under Operating System control.

(3) Space in fast store is allocated automatically for segments that need to be transferred from disk.

(4) A single address space is employed for the entire storage system. A segment of information is addressed by means of a unique reference, regardless of its location in the storage system. This reference is called a "Capability".

9. The System 250 Operating System contains two main components:-

(1) The Basic Control System controls the hardware configuration and provides a virtual machine for user programs.

(2) The User Terminal System employs the facilities of this virtual machine and provides a standard means of man-machine communication. In particular, the design of the User Terminal System is such that System 250 can be used as a time-sharing system for program development, including the development of large interactive systems.

BASIC MECHANISMS

Capability Registers

10. The CPU contains eight conventional, programmable Data Registers, and also contains eight programmable Capability Registers (Fig.2). The purpose of a Capability Register is to provide a means of addressing fast store. A Capability Register is 48 bits in length, and its format consists of four fields, namely a Store Module address, the base and limit addresses of a store segment contained within that Store Module, and an access field which defines the operations that are permitted on that segment.

11. CPU instructions access words within a segment by reference to a Capability Register which defines it. The address construction concatenates the Store Module and base addresses from the specified Capability Register, and using the resulting value as an addressing base adds the address offset and (optionally) a modifier value to give the absolute address of the referenced word.

12. The access field is a six bit code whose constituent bits define the operations that are permitted on the segment concerned, in any suitable combination. The "Read Data" and "Write Data" access types permit operations between the segment and the Data Registers e.g. Load Data and Store Data.

Similarly, "Read Capability" and "Write Capability" access types permit Load and Store operations between the segment and Capability Registers. The "Execute" type permits the segment to be executed as program code. The "Enter" type permits subroutine calls from one code segment to another 13. Thus, a Capability Register defines the permitted bounds of a segment and the operations permitted upon it. Any attempt by a program to refer outside these bounds, or to perform illicit operations, is detected by hardware in the CPU and the program is interrupted into a fault handler. The Capability Register can now be seen to have the following functions:-

(1) To provide an addressing base for access to segments in fast store.

(2) To protect a segment against illicit operations.

(3) To limit the scope of the program and thus protect the data structure outside this scope from illicit access.

14. Certain combinations of access types are not permitted in the system, as they would destroy the protection which Capabilities afford. For example, if an access field were to permit a value to be written into a segment as data and read out again as a Capability, then it would be possible to manufacture Capabilities and thence gain access anywhere in fast store. The general rule is that a segment is either of Capability type or of data type, but not a combination of the two.

Load Capability Instruction

15. Although there is a superficial resemblance between a Capability Register and a conventional base/limit register, there is a fundamental difference between them, namely that, unlike a base/limit, a Capability Register is alterable within user programs without recourse to a privileged operating system routine. Just as Data Registers are programmable, using CPU instructions like Load Data and Store Data, so Capability Registers are programmable using a separate set of CPU instructions, such as Load Capability (Fig.3) and Store Capability. There is a direct analogy between data and Capability types of information, Data and Capability Registers, data and Capability segments, and CPU instructions which either operate on data or on Capability values.

16. The ability of a program to load different values into Capability Registers permits it to gain access to as many different segments as is needful during its execution. At the same time, the scope of a program is bounded by the set of Capability values which its Capability segments contain.

17. The system would be very inflexible if a Capability value had the same format as a Capability Register; absolute addresses would then be scattered throughout fast store, and it would be extremely difficult to move the location

(3) No segment may be accessed except by means of a Capability. Capabilities therefore provide a single referencing system for the entire storage medium.

Structure of a Package

23. The Capability structure of a program package is depicted in fig. 4. It consists of a central Capability segment that defines a number of satellite segments, which may include further Capability segments. The central Capability segment defines at least one code segment and the data structure on which it is free to operate.

24. By hardware conventions, one of the eight Capability Registers (CR7) defines the code segment currently being executed by a CPU, and another (CR6) defines the central Capability segment of the package concerned. The code segment begins by having access to the central Capability segment, and as the code is executed it loads Capability values into the remaining Registers, in order to gain access to the required parts of its data structure. A jump to another code segment is achieved simply by loading the capability for it into CR7. Such jumps can, of course, only take place into accessible code segments.

25. If the package is a device handler for one of the devices attached to the CPU buses, then the device is accessed using a Capability which defines it. As the device handler is the only package that possesses a Capability for the device, the device is thereby protected from illicit access.

PROGRAM

The Enter Capability

26. In order for a package to call another as a subroutine, it must first possess a Capability for the called package. This Capability is of the Enter access type, which only permits a subroutine entry into a code segment of the called package.

27. Although the calling package possesses a Capability for the called package's central Capability segment, the Enter access type does not permit the Load Capability instruction to be performed, and the data structure of the called package is thereby protected.

Structure of a Program

28. A program consists of a number of program packages interconnected by Enter Capabilities as shown in Figure 5. The packages may be arranged in a conventional subroutine hierarchy, with one package being accessible from a number of others.

29. It needs to be emphasised that this program structure is established at assembly time. All the segments that constitute the program are assembled, and

of a segment, as this would require finding and changing all the absolute addresses which refer to it. Instead, all the base/limit values defining segments in fast store are collected into a single segment that is called the System Capability Table (SCT), which is defined by a non-programmable Capability Register. The format of a Capability value consists of the access field and an offset value to the entry in SCT in which the absolute location of the required segment in fast store is defined. The Load Capability instruction takes the access field from the Capability value and the remaining fields from the SCT entry.

18. In order to be able to move tne location of a segment, and to change the absolute values in its SCT entry accordingly, a facility exists to set a trap for programs which attempt to access the segment while these operations are taking place.

19. Inclusion of an access field within a Capability value, rather than in SCT, permits the access to a segment to vary from program to program. For example, one program may be permitted to read from a segment and to write into it, while another program is only permitted to read it.

PACKAGE

Capability as Access Right

20. In order to develop the concept of Capabilities further, it is necessary to dissociate it from the mechanics of addressing physical locations in fast store.This is achieved conceptually by replacing Capability Registers by Virtual Capability Registers, and thus by-passing the SCT. The Virtual Capability Register consists of an access field and a segment identity field. The Load Capability instruction can now be regarded as loading a Virtual Capability Register with a Capability value of similar form. When loaded, the Capability value (access right) permits the corresponding segment to be accessed and operated upon.

21. It was stated earlier that storage in System 250 is organised as a Virtual Memory. Once dissociated from fast store addressing, it can be seen how a Capability provides a unique reference to a segment in the Virtual Memory. It should be explained that the physical representation of a Capability on disk includes an absolute disk address, and that the Virtual Memory automatically converts Capability values from their disk form to their fast store form, or vice versa, whenever Capability segments are transferred in or out.

22. The concept of a Capability may be summarised as follows:-

(1) A Capability is an access right for a segment of store.

(2) The segment may be operated upon by suitable CPU instructions when the Capability is loaded into a Capability Register.

cross-references are resolved, prior to the first execution of the program. Capability segments are assembled in much the same way as code segments. In assembly language, a Capability consists of an access field and the symbolic name of the segment defined.

30. Within a program, two or more packages may possess a Capability for a common segment. Moreover, this may be a code segment. A good example is where there are a number of device handlers of the same type. Although each one is a package in its own right, they have an identical structure and their code segments are common. As each package possesses its own data structure, these code segments are in fact re-entrant. Indeed, as System 250 is a multi-processor the re-entrant executions could be simultaneous.

RESOURCE

Dynamic Allocation of Resources

31. The Operating System consists of a set of program packages which can be called in the normal way by means of an Enter type Capability. As the Enter type Capability provides protection to the called package, there is no need in System 250 for any privileged mode of operation.

32. One of these packages is called the Store Allocator. When entered during the execution of a program, the Store Allocator allocates a new segment and delivers a Capability for it. The segment may be a data segment or a Capability segment, and its length may be anything from one word to an arbitrary maximum of 1600 words. The Store Allocator is responsible for allocating space on disk and in fast store, for allocating a free SCT entry, and for putting into it the correct address values. The Store Allocator is the one place in the system where Capabilities can be manufactured.

33. During program execution the Store Allocator provides the storage resources required for the program to fulfil its task. A single store segment is, however, only the simplest form of storage resource, and much more complex data structures are possible. In a typical system, data structures of a particular form appear many times over. For example, in a telephone exchange system each data structure containing information about a subroutine may be of the same form. Other examples are a symbol directory in a time-sharing system, and an aircraft track in an air traffic control system. Given the basic facility of a Store Allocator, it is possible to envisage any number of resource allocator packages, each of which creates a data structure of some standard form.

34. The characteristics of a resource include:-

(1) A data structure, in tne form of an arrangement of Capability and data segments suitable for accommodating the data which describes,for example,

a particular subscriber or directory or track.

(2) The operations which are permitted on the data structure, e.g. to change a subscriber's class-of-service information.

35. The Operating System provides facilities for allocating and manipulating a number of resource types, listed as follows:-

store segment
synchronising flag
process
data stream
user
job
symbol directory
text file

Structure of a Resource

36. The structure of a resource, as created by a resource allocator package, is depicted in Figure 6. It consists of a central Capability segment with a number of satellite segments, which are the constituents of the data structure. This includes one or more code segments for performing operations on the resource. All the resources of a particular type have these code segments in common.

37. When a resource allocator package is entered during program execution it delivers a Capability for the newly created resource. Except in the case of the Store Allocator, this is an Enter type Capability which permits the constituent code segments of the resource to be called as subroutines. It is evident that a resource has the same structure, and is called in the same way, as a program package. The difference between them is simply that the resource is created dynamically during program execution. There is an important analogy between store segments and other types of resource. When a resource allocator package is called it creates a suitable resource and returns a Capability for it. In the case of a store segment, this Capability provides access rights to permit certain CPU instructions to operate on the segment. In the case of any other resource the Capability is of Enter type and provides access rights to permit certain code segments to operate on the resource.

38. The concept of a resource provides a method of creating protected data structures. Resources can be of arbitrary complexity according to a user's needs. A resource can be accessed by a single Capability which only permits a defined set of operations to be performed upon the resource.

PROCESS

Execution of a Program

39. During the execution of a program, a dynamic data structure may be constructed by repeated calls to resource allocator packages. This data structure is orthogonal to the program structure, and is related to the specific program execution. The term employed for the execution of a program is a "process", and this data structure is referred to as the "process data structure".

40. By a software convention, one of the Capability Registers (CR5) is used throughout a process to define the first Capability segment of the process data structure. As the process proceeds, Capabilities for resources that are created may be stored, by means of a Store Capability instruction, in this or subsequent Capability segments.

Call Return and Store Capability Instructions

41. As a process proceeds, it performs subroutine calls from one package to another by means of the Call instruction. The Call instruction has two parameters, the Enter type Capability for the called package's central Capability segment and an offset to the Execute type Capability of the code segment to be entered. The effect of the Call instruction is to load the Execute type Capability into CR7 and the Enter type Capability into CR6, in accordance with the conventions described in para. 24. Two further actions are required of the Call instruction:-

(1) In order for the code segment defined in CR7 to operate on the data structure defined in CR6, the latter must be supplied with a Read Capability access type. This is supplied automatically by the Call instruction.

(2) Before CR6 and CR7 are overwritten, their old values,together with the current value of an Instruction Address Register (IAR), are preserved in a stack (which is described below), so that these values can subsequently be restored by means of a Return instruction.

42. By convention, CR6 defines the package's data structure during its execution. As CR6 is overwritten by both the Call and Return instructions, this provides a further mutual protection of the called and calling packages' data structures.

43. The Return instruction is parameterless and its action is to restore the CR6, CR7, and IAR values for the calling package.

Process Dump Stack

44. Associated with each process is a segment that is called the "Process Dump Stack". It has two parts, as follows:-

(1) The stack area, used to preserve the CR6, CR7, and IAR values

during a Call instruction, by incrementing the stack pointer to define a new entry, which can then be used by the called package.

(2) A dump area, in which the remaining register values can be preserved on interrupt or context change.

45. The Process Dump Stack is defined by a special Dump Stack Capability Register. The preserving of a Capability value in the Dump Stack occurs each time that a Capability Register is loaded, i.e. at the time when the required Capability value is actually available. This takes place during a Load Capability instruction and also during a Call instruction, when the new CR6 and CR7 values are preserved in the new stack entry. Considering that these capability slots in the Dump Stack are loaded during a Load Capability operation, the location in the system of Virtual Capability Registers is now discovered.

46. The operation of the Store Capability instruction can now be explained. The action of this instruction is to store the value in a Virtual Capability Register into a word in a Capability segment. The Capability value is actually transferred from the corresponding Dump Stack slot.

Structure of a Process

47. A process is a resource. It possesses a data structure which is created by a Process Allocator package. A process may be created dynamically, e.g. the creation of a telephone call process when it is detected that a subscriber has raised his hand set, and in a time-sharing system the creation of a command interpreter process when a user logs in. Of course, the process must be created during the execution of another process, which must supply the Process Allocator with enough information to initialise a new Dump Stack e.g. initial CR6 and CR7 values plus any parameters that are to be set in the remaining Data and Capability registers.

48. The structure of a process is depicted in Figure 7. The central Capability segment of the process defines a number of segments which contain general information about the process e.g. its priority and the user job to which it belongs. One of the segments defined is the Process Dump Stack.

49. A process which creates another is supplied with an Enter type Capability which can be used to perform operations upon it, e.g. to block or unblock execution of the created process.

50. A process was described above as the execution of a program. It is better described as a data structure to which a CPU can be applied. From the Enter type Capability for a process stems an entire data structure which is activated and manipulated by the application of a CPU. From the CR6 slot in the Dump Stack stems the current program structure, and from the CR5 slot stems the active data structure of the process.

arbitrarily complex in its inter-connections; segments can be referenced from many different places, and Capability segments can refer back into limbs of the data structures of which they form a part. However complex the total data structure of a system becomes, it retains an essential order and discipline imposed by the mutual protection of its components and the systematic manner in which new components are created and cemented to the existing framework.

ADDITIONAL FEATURES

57. Finally, certain additional features have been identified which would enhance the benefits of a Capability system. Although these features are not yet implemented in System 250, they are included here partly for completeness and partly because the companion paper assumes them to be available:-

(1) Mixed Segments: A mixed segment is one which can include both data and Capability values. The introduction of mixed segments removes the rigid distinction between data and Capability segments and therefore provides greater flexibility in the structuring of information. In order for there to be no loss of protection, the distinction between data and Capability types of value attaches to the values themselves.

(2) Process Workspace Stack: The purpose of this stack is to supply a package automatically with working space when called during the execution of a process. This working space is referenced relative to the stack pointer. The stack may also be used to preserve and protect a package's working data, when a further package is called, by incrementing the stack pointer by a suitable value.

SYSTEM

Change Process Instruction

51. In a system there may be many processes simultaneously in existence. In terms of a virtual machine (para.9), each process has a virtual processor on which to operate. The mapping of virtual processors onto a smaller number of real CPUs is achieved by queueing the processes for an available CPU. When a process occupying a CPU either ends or arrives at a suspension point, it enters a CPU Scheduler package. The Scheduler selects the first process from the queue and performs a context change into it, so that this process now occupies the CPU.

52. A context change is achieved by the Change Process instruction. This instruction has a single parameter, namely a Capability for the Dump Stack of the new process to occupy the CPU. The effect of the Change Process instruction is to store the current Data Register values in the dump area, load the Dump Stack Capability Register with its new value, reload all Data Registers from the dump area, and reload all Capability Registers, including the values in the current entry of the stack.

53. The execution of Change Process transfers complete control of a CPU from one process to another. The effect of an interrupt on a CPU is to force a Change Process from the one which currently occupies the CPU into the interrupt process. As the context of the old process is preserved in its Dump Stack, control can subsequently be restored to it.

Structure of a System

54. This paper has described how everything in a system, even the representation of processing, is reducible to data structures. It has also described how the data structures which comprise a system are built up from component data structure, an inter-connected network of Capabilities providing the cement which binds the whole structure together.

55. A system is constructed from static data structures, including preset data and code segments, packages, and programs, and dynamic data structures, including processes and other resources. Preset data and code are assembled and loaded into the system, where they reside in segments of the Virtual Memory. Packages are formed by assembling and loading Capability segments defining their constituent segments. Programs are formed by the inclusion of Enter type Capabilities within these Capability segments. Processes and the resources they create are formed when CPUs are applied to these static program structures.

56. The resulting system is an ever changing list structure that can be

REFERENCES

1. Cosserat D.C. A data model based on the Capability protection mechanism. Proceedings of the International Workshop on Protection in Operation Systems (this proceedings), IRIA, Paris, August 1974.

2. Dennis J.B. and Van Horn, E.C. Programming Semantics for multi-programmed computations. Comm. ACM 9, 3, March 1966.

3. Wilkes M.V. Time sharing computer systems. Macdonald. London 1968.

4. Fabry, R.S. List - structured addressing. Ph.D Thesis, University of Chicago, 1971.

5. Fabry, R.S. The case for capability based computers. Presented at Fourth Symposium on Operating System Principles, Yorktown, New York. Oct. 1973.

6. England D.M. Architectural Features of System 250. Infotech State of the Art Report 14 - Operating Systems, 1972.

7. Papers by Cotton, J.M., England, D.M., Halton, D., and Hemmings W.A.C., in Proceedings of International Switching Symposium, Boston, Mas., June 1972.

8. Papers by Crompton J.M., Cosserat D.C., Hamer-Hodges K.J., and Repton C.S., in Proceedings of International Conference on Computer Communication, Washington, Oct. 1972.

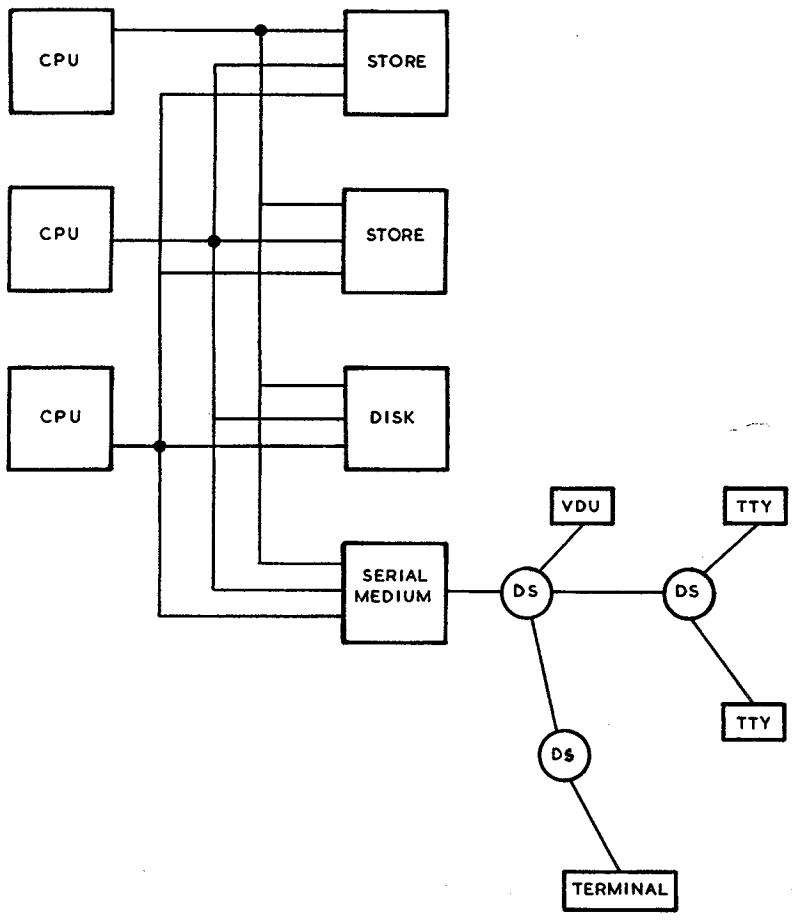
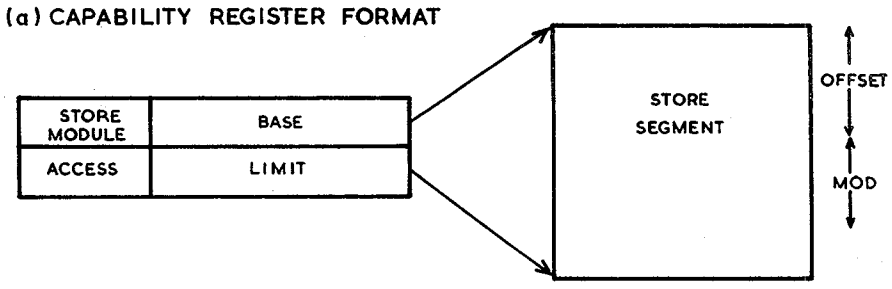
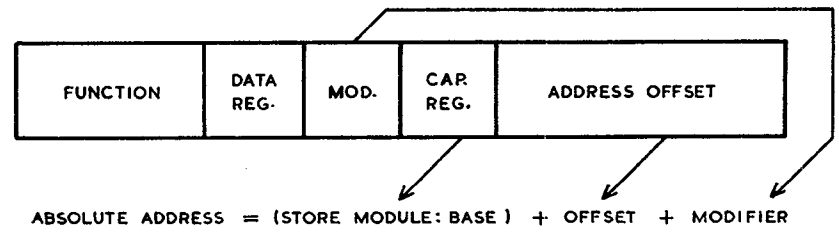


FIG.1. SYSTEM 250 - TYPICAL HARDWARE CONFIGURATION

(a) CAPABILITY REGISTER FORMAT



(b) INSTRUCTION FORMAT & ADDRESS CONSTRUCTION



(c) ACCESS TYPES

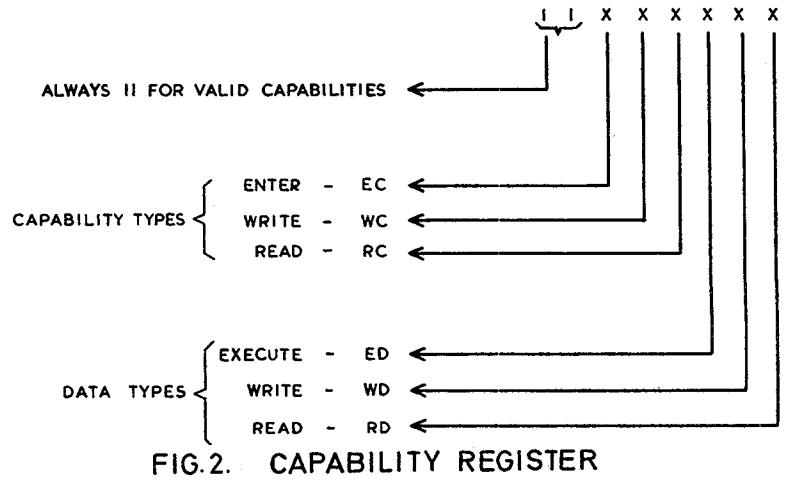


FIG.2. CAPABILITY REGISTER

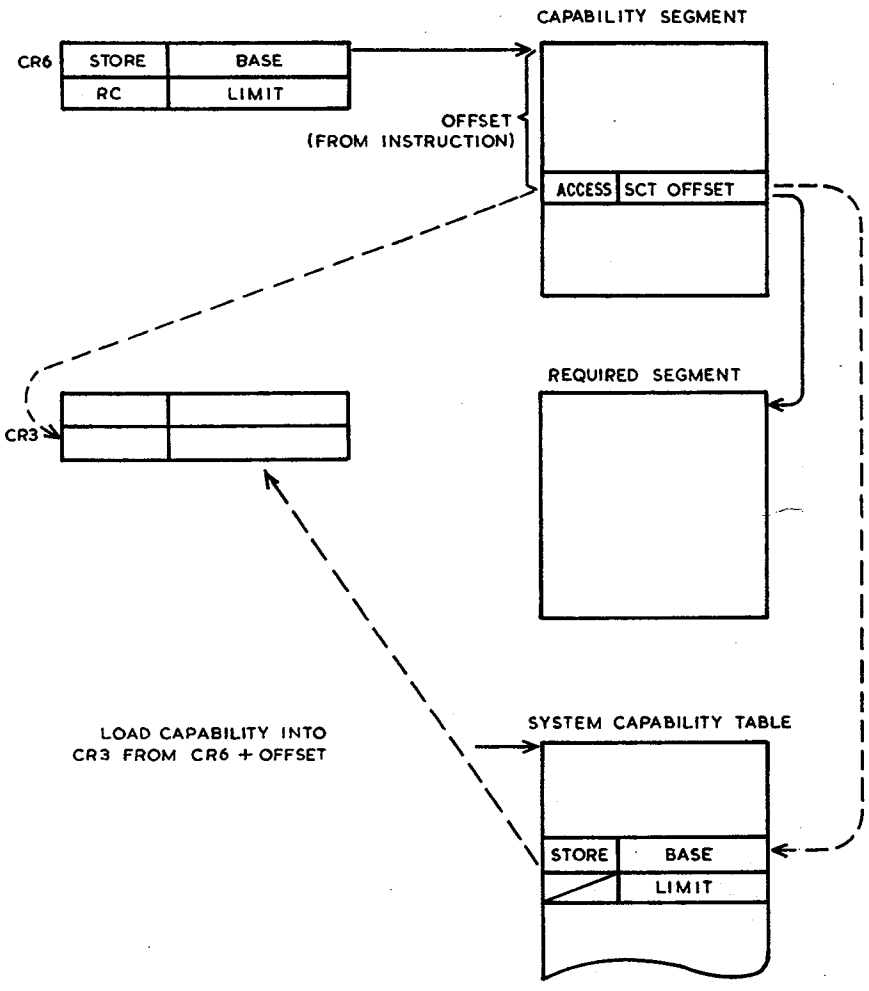


FIG.3. LOAD CAPABILITY

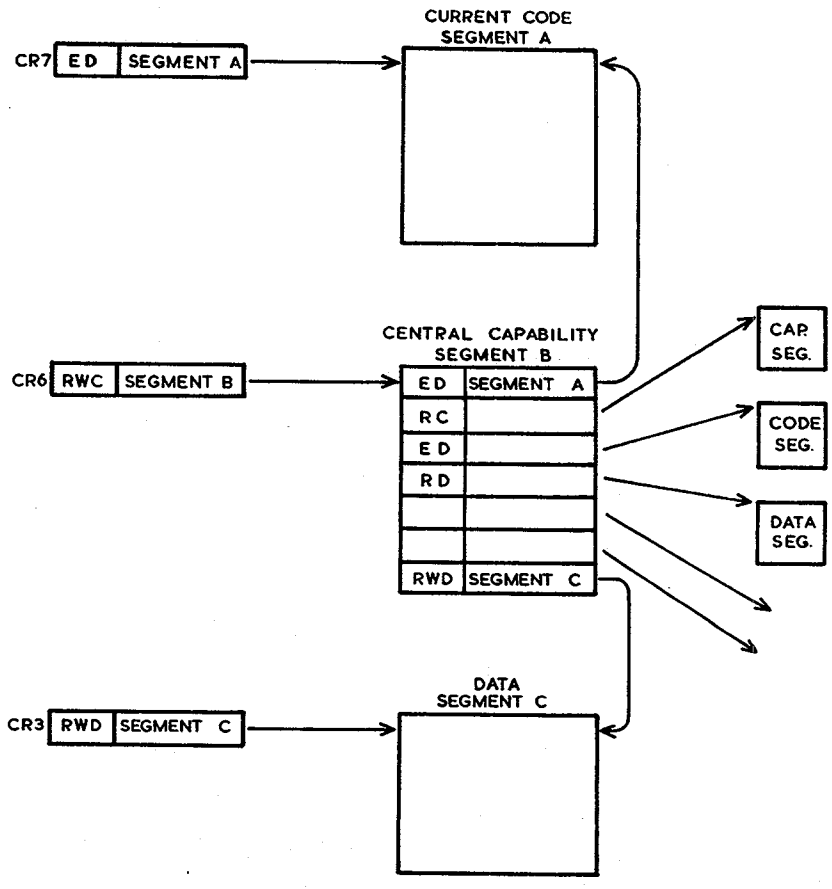


FIG.4. STRUCTURE OF A PACKAGE

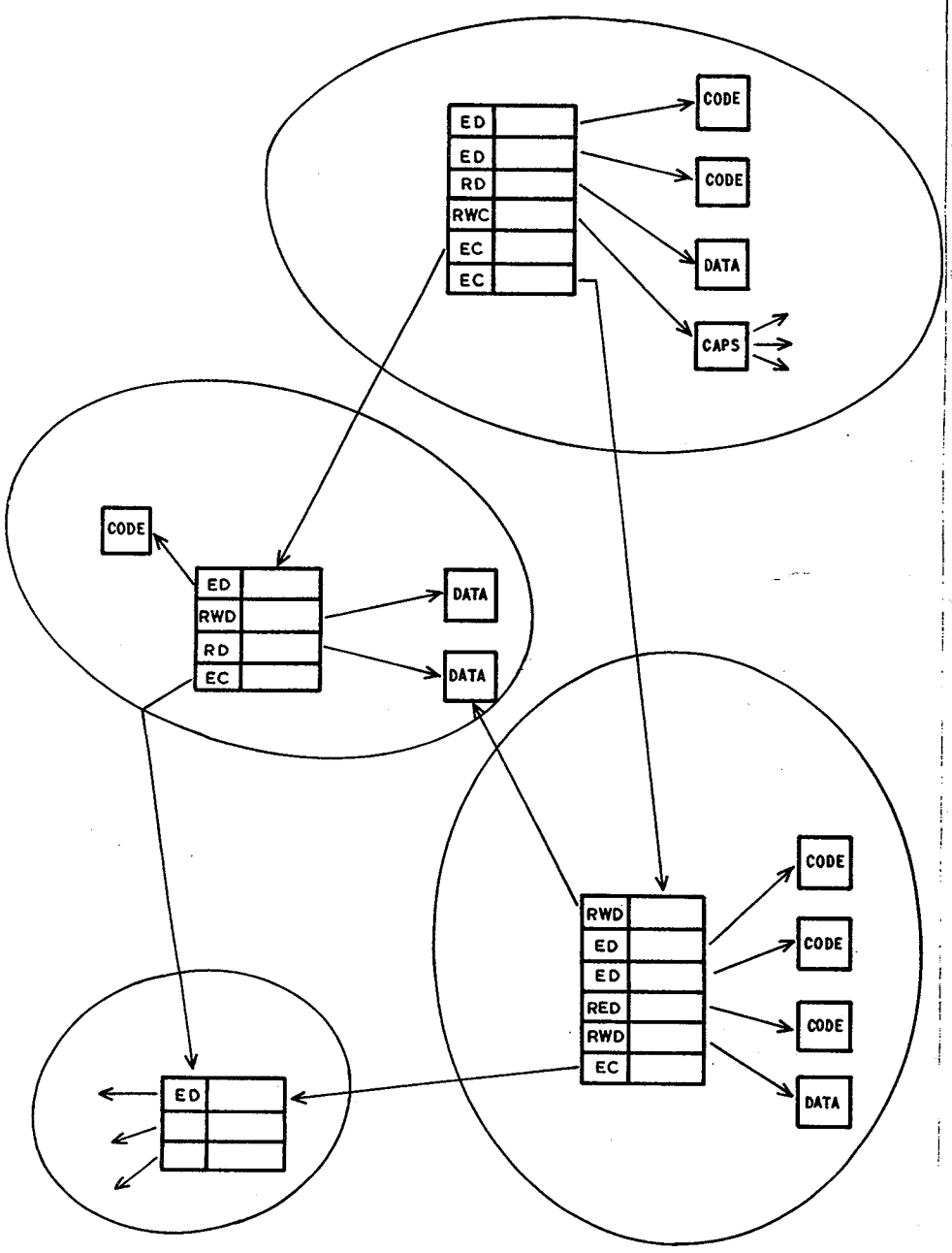


FIG. 5. STRUCTURE OF A PROGRAM

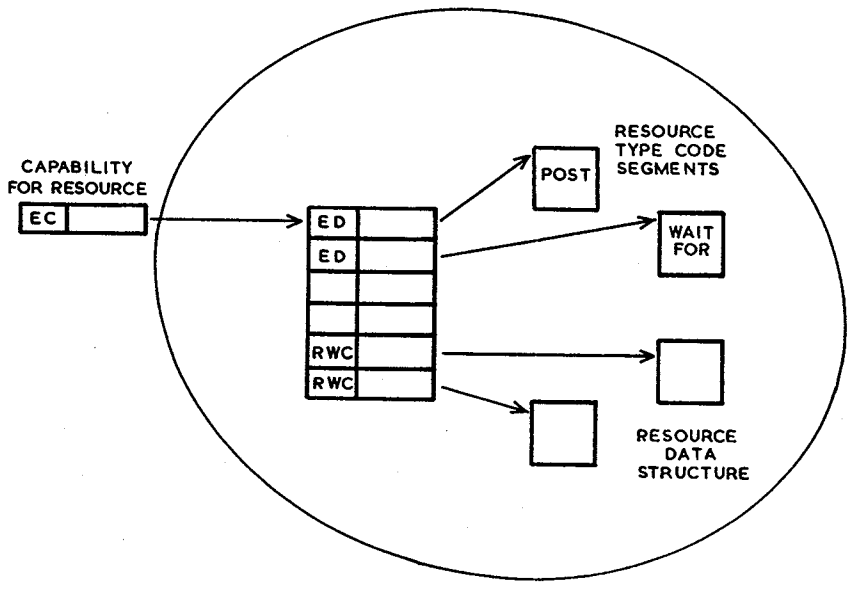


FIG. 6. STRUCTURE OF A RESOURCE
(SYNCHRONISING FLAG)

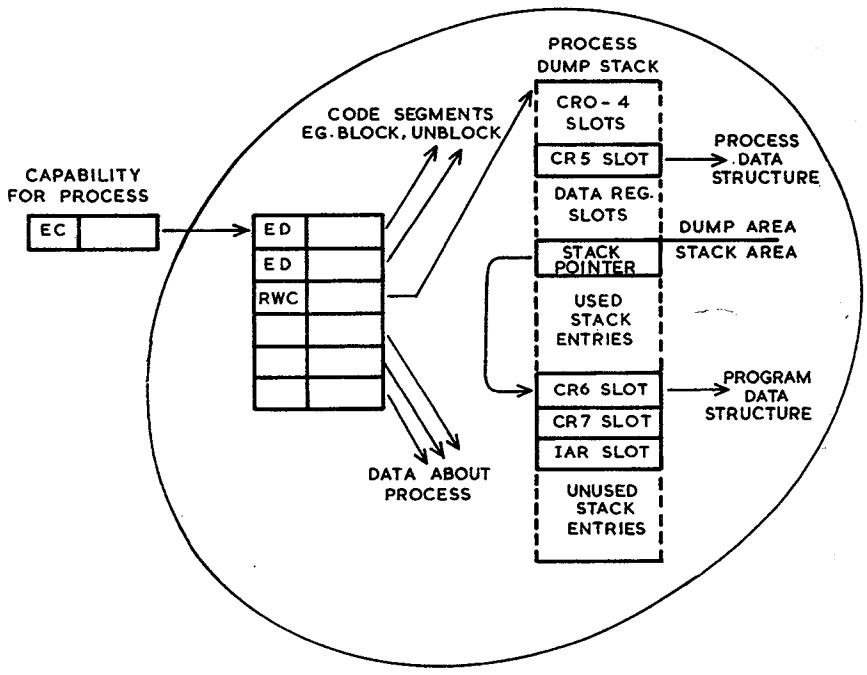


FIG.7. STRUCTURE OF A PROCESS

AN EXTENSIBLE STRUCTURE FOR PROTECTED SYSTEMS' DESIGN

J. FERRIE
C. KAISER
D. LANCIAUX
B. MARTIN
IRIA/LABORIA, Rocquencourt, France

PRELUDE

This note is the first draft of a study being developed at IRIA on the topic of protection related to operating systems. The purpose of this research is to develop and analyze addressing and protection structures which will help to run more secure programs and to limit the propagation of errors. We are planning to microprogram these structures on a minicomputer and to implement a small but significant subsystem.

INTRODUCTION

Statement of the problem

Protection is the general term used to describe the action of hardware and software mechanisms which control the access to the items of a system. Its purpose is not to avoid the production of errors, but only to prevent their propagation towards protected items [Crocus]. In the present article, system security which determines the user's confidence in the system and which is based on the correct use and reliability of the protection mechanisms, is not taken into account.

Systems items which may generate errors and whose actions have to be controlled are called agents. A process is an exemple of an agent. Items which require protection are referred to as objects. An object is defined by the operations that can be performed on it. Protection mechanisms will not allow us to operate on an object with unpermissible operations.