ISE-TR-86-59

A LOOSELY COUPLED MULTIPROCESSOR SYSTEM : ADMS

—— LOGICAL CONFIGURATION & OPERATING SYSTEM ——

by

Sanae AMADA

Yutaka SATO

Syu-ichi SUZUKI

October 16, 1986

INSTITUTE
OF
INFORMATION SCIENCES AND ELECTRONICS

UNIVERSITY OF TSUKUBA

A LOOSELY COUPLED MULTIPROCESSOR SYSTEM : ADMS
---LOGICAL CONFIGURATION & OPERATING SYSTEM---

by

Sanae AMADA*, Yutaka SATO**, and Syu-ichi SUZUKI***

* Institute of Information Sciences & Electronics,
University of Tsukuba.

** Doctral Program in Engineering,
University of Tsukuba.

*** Intel Japan Co. Ltd.

Abstract

We have described on the basic design of ADMS, already |Ama86|. In this paper, we add some results of our works concerning with the logical configuration and the operating system of ADMS to the previous report. The reasons of use of the capability-based addressing, the addressing mechanism, the executive environment of the program, the object oriented environment, and the basic configuration of the operating system are discussed.

Contents

1. INTRODUCTION

ADMS has an object oriented architecture, and its object corresponds to the model of objects as encapsulated data in the definition by Ishikawa and Tokoro |Ish84|. They said on needs of increasing of the parallelism in this model. ADMS has the high parallelism by introducing of the concept of the process.

As well known, in an object oriented architecture, the operation of an object is restricted to a procedure defined with the type, so we can have enough protective ability with this architecture. And further, we use the architecture because of it can fit for the loosely coupled multiprocessor configuration.

Because we made every effort to hide the multiprocessor configuration to the user, ADMS could have good and easy interface for the user.

In following chapters, we describe on the logical configuration and the operating system of ADMS. The total image of ADMS is described in |Ama86|, already.

2. CAPABILITY-BASED ADDRESSING

We have to use logical addresses which is independent to the physical system configuration, and we apply a capability-based addressing method for our system. This method is studied for the flexible protection |Fab74| and applied on commercial systems |Hou81||Tyn81|, recently. We use this method not only for the protection but to realize an enough fitted addressing method for the system.

The 1st reason

The capability can be used as an unique address for any resource in the system. In a loosely coupled system, we must access a target which exists in another physical addressing space, or an addressing space belonging to another processor. The capability can be a logical unique address, so we can construct one uniform logical addressing space with it on the system. This construction will have following features.

1. A caller has no influence by the migration of a target among the processors.
2. The execution of a caller itself have no restriction about the migration of it among the processors. This will bring advantages for load balancing and the reliability of the system.
3. Even if we change the hardware configuration, there is no effect on the construction of the logical addressing space. This means easy extension or reconfiguration of the system.
4. We can describe the software independent of the hardware configuration. It gives more easy programming.

The 2nd reason

With the set of capabilities or the capability list, we can define and control all executive environments on the system. Here, the environment means a set of access rights. By applying of this method, "principle of least privileges" |Lin76| can be realized dynamically, truely, and with the high efficiency.

The 3rd reason

In "right-list method" the access right is checked at

the object side after the access has occured. In "capability method" the access right is checked at the subject side in the first stage as the resolution of the object name occurs. In the loosely coupled system, there will be a status that the subject exists separated from the object physically. So, the feature of the capability method, in which the check is completed without accessing on the different space, is important in a viewpoint of efficiency.

To implement a capability-based addressing mechanism, following items must be studied.

1. The generating method of the unique address in the system, and the guarantee method for the uniqueness.
2. The method to convert a logical unique address into a physical address on each processor.
3. The control of access rights in a fine-textured and high-efficient manner.
4. The method to lighten the overhead caused by referring to the capability list.
5. Problems caused by the fact that we have no way to notify of the change of status of any object to the subject.

To solve these problems we selected following mechanisms. The reference mechanism to the capability is woven into the addressing mechanism as the hardware. As the convert mechanism of the logical address into the physical one, a newly designed hash mechanism is used, and it concerns with the communicating function via busses. Besides, the generating function of the unique address and keeping up of the uniqueness of the address are left to the hash mechanism. To realize these functions, a new distributed addressing system

is introduced. To increase the flexibility of the protection and the writability of the program, two special capabilities are prepared. Details are described in following chapters.

3. ADDRESSING MECHANISM

The first problem on addressing is the decision of the logical size addressed by a capability. Though there are many terms to be discussed for the decision, we select as the size as a segment which contains a series of 64 KB area on the physical storage space. That is, a storage space is a set of segments, and each segment is addressed via capability. The protection of the smaller sized area is realized by a special capability.

We put the information of attribute of segments in the segment descriptor. By a viewpoint of efficiency, the information of attribute ought to be put in the capability. But, in our system, the information which may be changed after the creation of the segment can not be put in the capability. The reason is ; plural capabilities can point one segment and we have no way to find a capability which points a specific segment. This configuration brings the relatively short capability in its length.

The segment and the corresponded segment descriptor are placed in a porcessor, and the latter is in a local segment table. Meanwhile, to prevent unreasonable accesses on the capability, we prepare C (capability) segments and D (data) segments. The configuration will bring us the enough protection of capabilities and the high efficiency on the operation to executive environments and addressing.

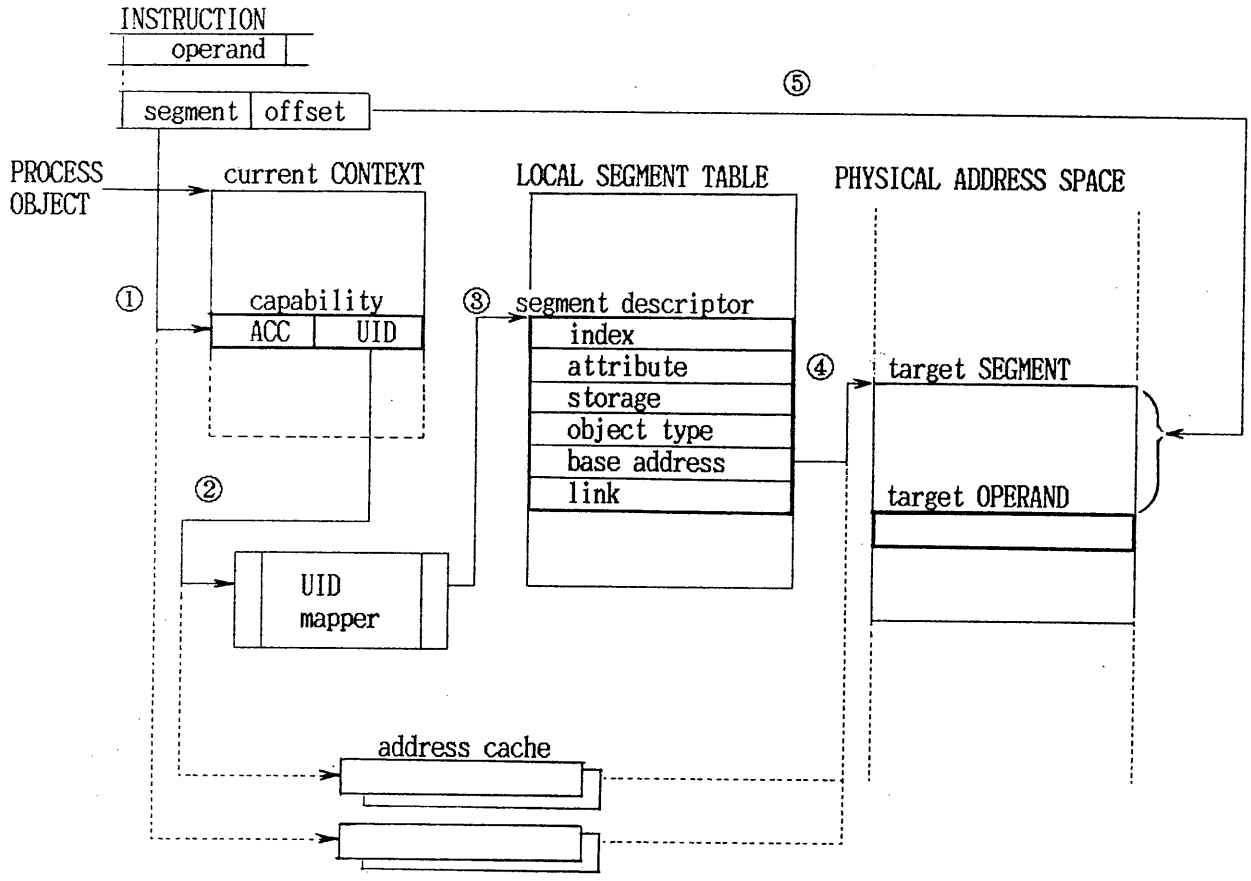

Fig.1. The Address Conversion Mechanism.

In Fig.1, we show an address conversion mechanism on each processor. In the figure, we get a segment designation part and an offset designation part from the operand part in the instruction. The check of access right and attribute of the object is executed in the mechanism. For instance;

in route 1 : comparing of contents of the instruction and the access right written in the capability, and checking of the mode bit in the capability,

in route 3 : comparing of contents of the instruction and the segment type in the segment descriptor, and

in route 5 : comparing of offset value and the segment length in the segment descriptor.

To lighten the overhead of the addressing mechanism, we pre-

pare two associative address caches, and implement the hash mechanism and segment table with hardware elements.

In the UID (unique identifier) conversion mechanism, we have functions to prevent the duplication of the UID and to search segments all over the system.

3.1 Creation and Destruction of the Segment

To create the segment, we have to generate an UID and register a segment descriptor on the segment table. To destroy the segment, we have to abandon an UID. We put an UID generating counter on each processor and promise to express the processor number with higher 4 bits of the UID. Of course, each segment can travel between processors after the creation.

At the creation of the segment, by checking of the collision in the hash mechanism we can confirm that the prepared UID is out of use. At the destruction, by testing with the count of the capability we can check whether the said UID is in use or not. Thus, we can keep the one to one relationship between the UID and the segment. The count of the capability is renewed or tested by copying or destroying procedure of the capability.

In our algorithm, if the collision has occured by hashing, we repeat to generate the UID until we can find any empty entry on the table. We prepare a "segment exist bit" in the segment descriptor and keep said entry as "valid" till the capability which points said segment has deleted. The test of count of the capability and the deletion of the entry will be executed in the deleting procedure of the capability.

In the algorithm, there will be no collision in hashing at the reference. Because the collision is avoided at the generation in advance. As a result, though the overhead at the generation is pretty large, the total overhead will be small by the reason that the reference will occur much more times than the generation.

In addition, we separate UIDs as parmanent and temporary ones. The number of generating times and the life time of them will be compensated each other, and it will reduce the collision.

3.2 Import and Export of the Segment

On the occasion of registering of an imported segment, we cannot apply the previous mentioned algorithm. We must apply a collision management just same in case of the ordinary hash mechanism. When we have a collision, we put said segment descriptor on a link-storage and link it to same hash-valued segment descriptor.

It must be noticed by the export that we cannot rub off the segment descriptor of the exported segment. That is a mark to show that the corresponded UID is in use. To show the absence of the segment, we turn off a "local bit" in the segment descriptor.

In our system, we transfer data in not the segment form but the object form.

3.3 Reference of the Segment

As a basic search procedure, we have the "local_segment_search" procedure.

In a case of the capability is transfered from another

processor or the segment is exported, there may be no target segment in own processor. Then, a procedure which asks some operations to other processors is started. By the start, the control of the execution is transfered to another processor. In our system, a processor does not contact with the physical storage spaces of other processors, and the control is limited to the operation of the object and the system procedure.

An asking procedure "external_request_send" is a procedure which transfers a message to other processors. By receiving of the message, the "external_request_serve" procedure starts its function containing the "local_segment_search" procedure. At first a processor which has created the target segment is selected. We are able to know the processor with higher 4 bits of the UID. If the target segment cannot be found in the processor, the request is sent to plural processors at a time by a broadcast function prepared in the BIU (bus interface unit).

3.4 Garbage Collection

In ADMS, when the count of the capability in the segment descriptor come to zero, the segment descriptor is rubbed off and the storage area of the segment is released.

3.5 The form of the Capability and the Segment Descriptor

```
CAP : record
        status      : record
                        valid      : boolean;
                        mode       : capability_type;
                      end record;
        self_right  : record
                        delete_right  : right;
```

```
                    copy_right     : right;
                    move_right     : right;
                  end record;
    access_right: record
                    right_1   : right;
                    right_2   : right;
                              .
                    right_n   : right;
                  end record;
    uid           : id;
  end record;

/*  boolean   ={ on, off}
    right     ={ on, off}
    capability_type  ={base, abstract}
    id        ={ 0 ... maxuid}

SD  : record
      index     : record
                    valid     : boolean;
                    d_type    : descriptor_type;
                    uid       : id;
                  end record;
      storage   : record
                    exist     : boolean;
                    local     : boolean;
                    move      : boolean;
                    limit_flag  : boolean;
                    cap_limit   : ordinal;
                    cap_count   : ordinal;
                  end record;
      attribute: record
                    s_type          : segment_type;
                    segment_length  : ordinal;
                  end record;
```

```
        base_address   : ordinal;
        object_type    : id;
        link_pointer   : record
                           link_flag    : boolean;
                           link         : ordinal;
                         end record;
      end record;

/*      descriptor_type = {SD, RD, ID}
        segment_type    = {Capability, Data}
        ordinal         = {0 ... max}
```

3.6 Special Capabilities

A refinement (a serial part in a segment) is defined as a virtual segment and a capability is given to it. Information of the mother segment and the refinement itself is described in the refinement descriptor. The refinement descriptor is treated just same as the segment descriptor. The access path is shown in Fig.2.

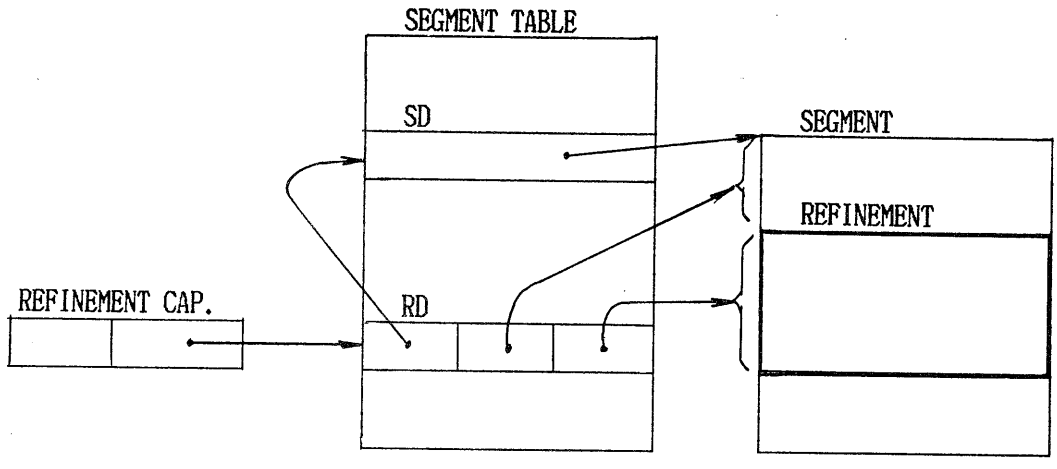

Fig.2 The Access Path of the Refinement Capability.

An indirect capability is prepared, too. This is effective for the selective cancellation of the transferred capability and the dynamic indirect operation of the executive

environment |Mye80|. The access path is shown in Fig.3.

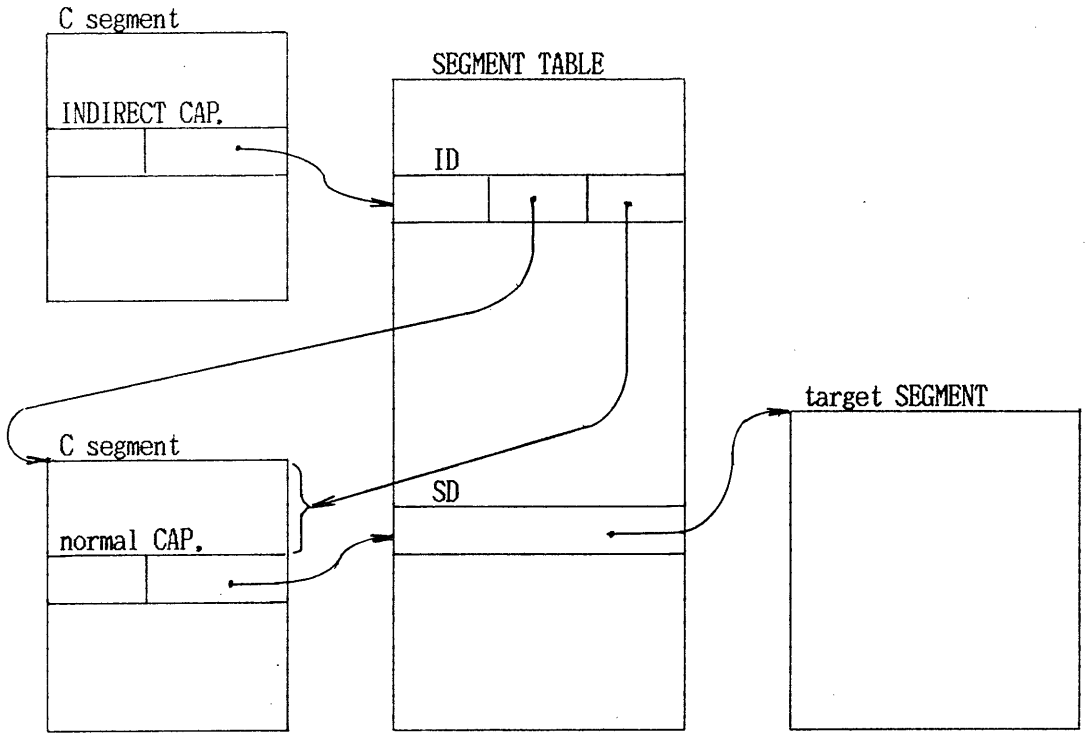

Fig.3 The Access Path of the Indirect Capability.

3.7 Operational Procedure on the Capability

DELETE, COPY, and MOVE procedures are prepared for the capability, and flag bits in the self right field of the capability are corresponded to them. At the creation of the capability, all flag bits are set in "on", and controlled by the RESTRICT procedure afterwards.

A procedure used to increase self and access rights is AMPLIFY. At first, "amplify control object" is created by type managers. And this object has a cast of right field of the capability. The AMPLIFY procedure receives amplify control object as an argument and executes "or" operation on the right field of the target capability. Thus, the AMPLIFY procedure is a privilege operation of type managers.

The RESTRICT procedure receives "restrict control object " which has a same configuration with the amplify control object, and executes "and" operation on the right field, All subjects can create this object.

We have "set capability limit" procedure to set the upper limit of number of capabilities corresponded to one segment.

4. EXECUTIVE ENVIRONMENT OF THE PROGRAM

4.1 The program Module

In ADMS, the executive environment is defined as the set of capabilities. An address space shown by this set is called as a domain. We have three system object types to define the program module.

INSTRUCTION is a D segment composed of a set of machine instructions and the heading containing control information. This is an entity which defines one sub-program, and is prepared to protect the series of instructions from undesirable operation. Contents of it can be changed only by a renewal operation defined by the system.

DATA is a D segment which has no types. Contents are static variables in the program and constants generated by the compiler.

DOMAIN is a C segment composed of a set of capabilities, and shows an access space corresponded to a static executive environment of the program. And this defines a program module putting together instruction objects, data objects, and other typed objects in logical. The inner format of it is

defined by the compiler.

Fig.4 shows the relation of these three types of object.

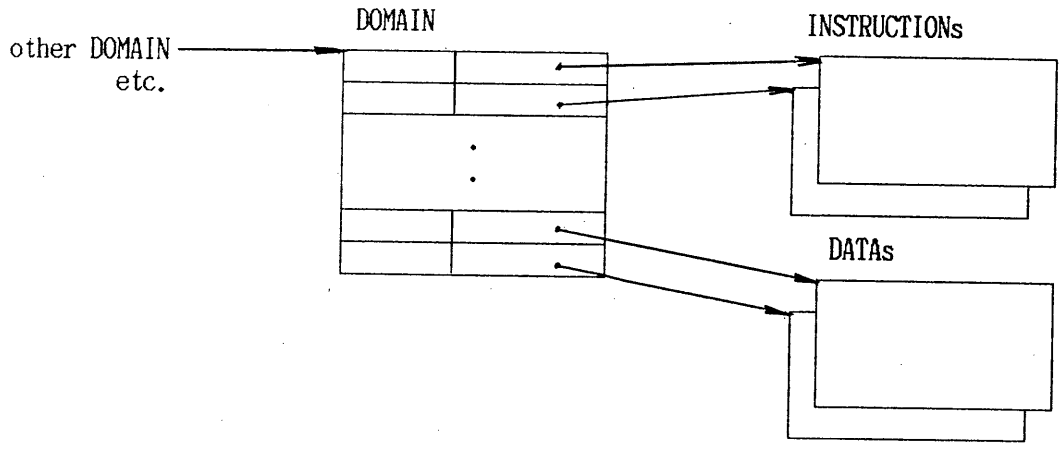

Fig.4 The Definition of the Program Module.

The accessability into other modules is shown whether the own domain has a capability for the domain representing the target module or not. Of course, the capability must have a proper access right. For the access right of a part of module, a refinement capability is used. To control the target after the transfer of the access right, an indirect capability is used.

To keep higher efficiency, it is preferable that links between a domain object and instruction and data objects are not ranged over from processor to processor.

4.2 Control of the Executive Environment

To define the dynamic executive environment of a program module, we prepare a system object type named context. The context object is created dynamically by the call of the sub-program and linked up to the old context object. Thus, the change of the executive environment of programs is expressed as a linear list composed of links between context objects,

and the last context object shows the current executive environment.

A context object is composed of from 2 to 4 C segments (CCS0~CCS3) and a D segment (CDS). CCS0~CCS3 are set in the capability list in the address conversion mechanism. Namely, capabilities in CCS0~CCS3 define the sets of physically accessable targets.

The designation part for the segment in the instruction has following form.

<segment_selector> ::= <CCS_number> < index >

The construction of the context object and the path to designate segment is shown in Fig.5.

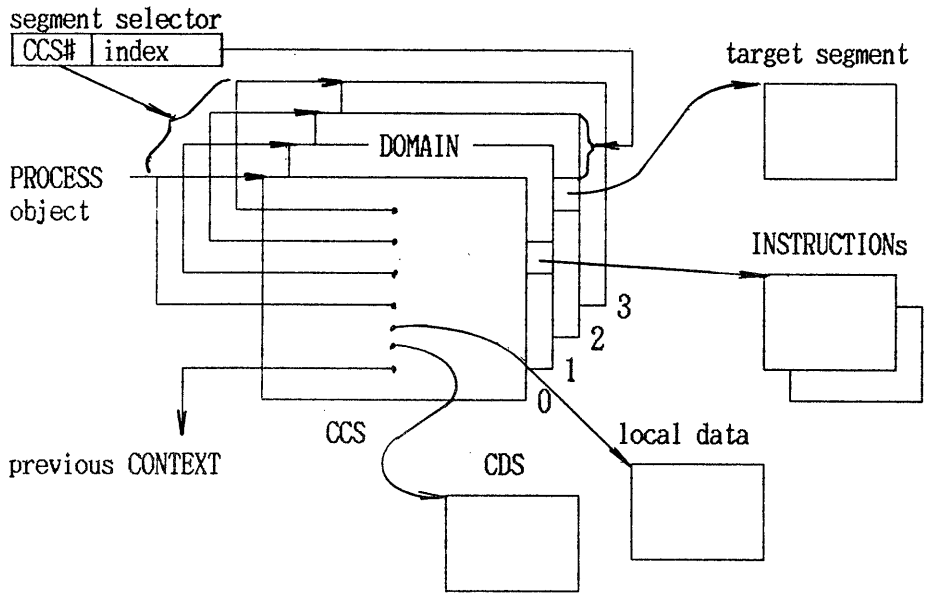

Fig.5 The Configuration of the Context.

CCS0 contains capabilities for links and has a role as a root segment (CRS). We put the domain object on CCS1. CCS2 and CCS3 are prepared for the indirect access. For the indirect access operation we have ENTER_CONTEXT procedure and REMOVE_CONTEXT procedure. CDS is a stack for storing of the program status block (PSB).

4.3 Change of the Executive Environment

Following three operations are prepared for the transportation of the control among sub-programs, and shown in Fig.6.

BRANCH is the simple transportation of the control among instruction objects. PSB must be changed by BRANCH operation. This is used to link the multiple instruction objects those show large sized sub-programs as over 64 KB.

LCALL/LRETURN is used for the call/return of procedure of the instruction object in current domain. By the operation, "push/pop" of PSB, local variable data, and local stack data in CDS are carried out.

CALL/RETURN is used to call the procedure in the other domain and takes with context switching.

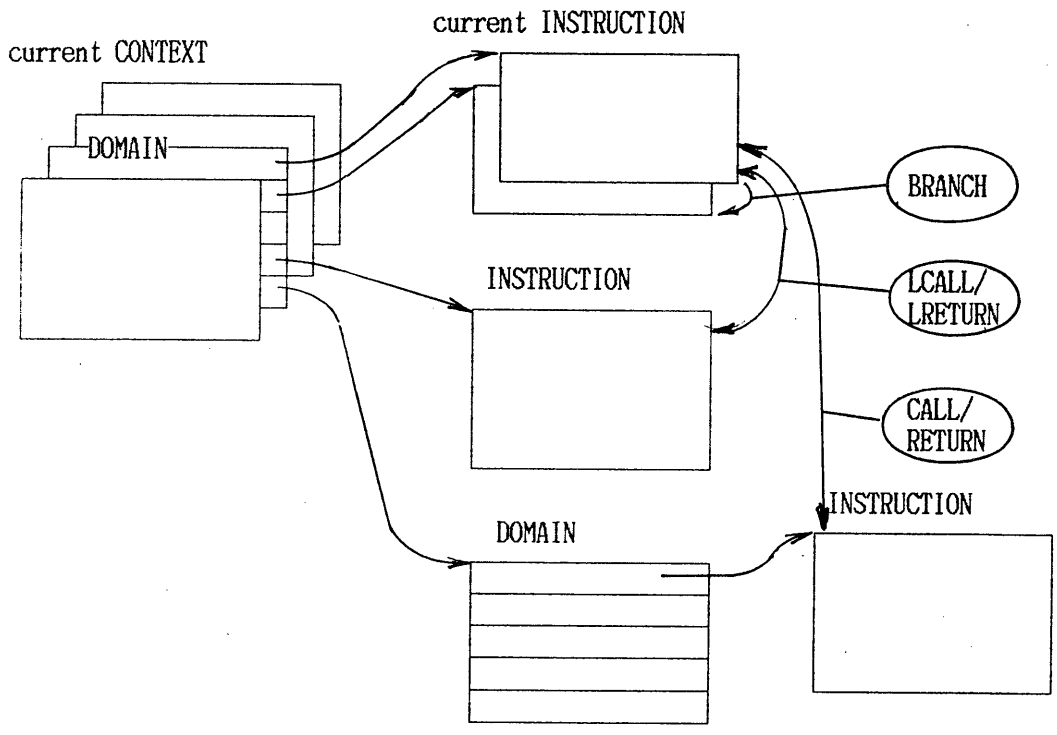

Fig.6 The Transportation of the Control.

5. OBJECT ORIENTED ENVIRONMENT

In ADMS, we construct the object oriented environment in which all resources in the system are defined as abstract entities having unique names and types, based on the addressing mechanism and the control mechanism of the executive environment as mentioned above. Here, an object is a passive entity, and the internal construction of it is hidden by the type.

5.1 The Definition of the Object

An object is composed of a C segment and a D segment, or a hierarchical construction linked with a C segment. The C segment is called as a root segment.

We call a capability for a root segment of an object as "a capability for the object", and use for addressing to the object. A capability for the object is defined as an "abstract mode" capability distinguished from a basic mode capability. The latter recognises the target as the physical segment. To the former, it is applicable only an abstract operation defined by the type manager.

The type of the object is defined by the information of the object type field in the segment descriptor of root segment. A form is usable which contains ones defined as object previously, instead of the segment. The refinement is applicable.

As mentioned already, we avoid the distribution of one segment construction to plural processors. This is able to realize by the restriction of the transportation of data between processors in a unit of object.

5.2 Accessing to the Object

Accessing to the object is started by sending message from the accessor to the type manager which defines the type of the target object. The type manager operates the object directly according to the received message. Said object and the type manager must be in existence on a same processor.

The message includes a capability for the object, a name of the operation and arguments. The type manager send back the result of the operation to the accessor. On the object of the system type, there is no type manager.

5.3 The Configuration of the Type Manager

In ADMS, a type manager is defined as an object with type form, and treated equally to general objects. The reason of this adoption is as follows.

* In our system, the expandability and the easiness of creation of the type are attached great importance. By the adoption, we can create types with ease and without any restriction in number.
* A complex configuration as hierarchical form and so on is realized easily.
* The protection mechanism is applicable for the type manager just same to objects.

The type form is one of the object type offerd by the system, and its type manager (type-type manager) is woven into the system. The type-type manager prescribes the configuration of the type manager, and has a role to create the types.

The internal configuration of the type manager is copied

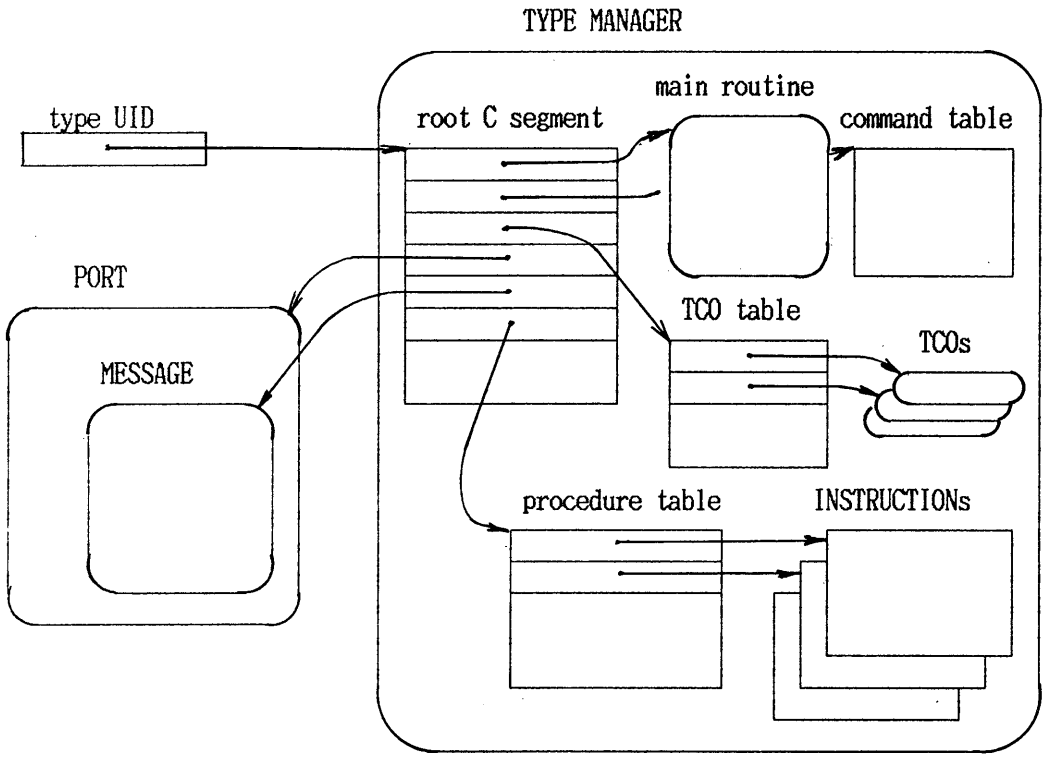

Fig.7 The Configuration of the Type Manager.

from the type definition of the language, and shown in Fig.7. The main routine of it is composed of a message analyzer and a routine which checks type rights. The type manager receives the message and analyze it using the command table and TCO (type control object) table, and searches the corresponded procedure in the procedure table, and then starts.

5.4 SEAL/UNSEAL Function

SEAL is a procedure to set the type for the set of segments, and UNSEAL is a procedure to release the type for the object. And, TCO is an object to qualify for the start of procedures. At the time, we must designate TCO as an argument. TCO is given to the type manager from type-type manager at the creation of the type, and it is an independent object by the reason of the protection. SEAL/UNSEAL procedures of

types are privilege operations of type managers by owing capabilities to TCO in type managers.

SEAL/UNSEAL procedure send back the capability which contains proper access rights. If needed, the capability is converted to the abstract mode by SEAL, and is converted to the basic mode by UNSEAL.

If we try to apply a procedure call mechanism used in Hydra|Wul81|, iAPX432|Tyn81|, etc., we meet some ploblems. In these systems, the procedure call mechanism is based on the sequential processing mechanism. So, by the application of this mechanism, the parallel processing ability by plural processors in ADMS cannot be utilized. And then, if an object and a corresponded type manager are distributed in two processors, the efficiency of the execution cannot be kept in high level. Thus, we prepared following new mechanism. In the new mechanism, a type manager is executed as a process, and an access from accessor to object is executed as the exchange of the message between an accessor process and a type manager process.

We prepare REQUEST procedure to send messages, and the procedure has the object level message as the argument. In the procedure, the object level message is converted to the process level message. And then, the message communication between processes are started. We will write on the message communication in chapter 6.

As a result, the system environment is constructed of the set of resources defined as objects and the message passings between resources, in logical. In physical, they are the set of processes and the message communications between

processes.

Thus, all physical active entities are defined as processes, and have an unified interface. The process is the basic unit of concurrent programs, and an accessor process and a type manager process can be operated independently and in parallel.

5.6 Access Management

Each processor has a TPT (type process table) which is a set of type descriptors, and UID of the type manager is registered onto it at the creation of the type manager. Then, to some processors, they may use said type, the registration is carried out by broadcasting. This configuration makes the easy dynamic system reconstruction.

The access to the system type object and other type object are equally started with REQUEST procedure, but the progresses are somewhat different naturally. By the access to the object on other processors, roughly speaking, the same progress is started after the search operation over the system.

5.7 The Harmonization with the System Configuration

Type managers are managers of resources, and so the operating system of ADMS is the set of type managers. In our system, resources are distributed in response to the function of each processor, and the management of each processor is satisfied by the distribution of type managers. If we are needed to set the same function on plural processors, there will be no problem by the distribution of the same type manager. For the management of the status, a system type table

is prepared in a system control processor (PU #0).

This configuration will have a result of load distribution by itself. In the case, to improve the efficiency, we set up following rules.

* When the target type manager exists on own processor, that type manager is selected.
* If not, request messages are transmitted to all or specific processors by the broadcast function.
* The first arrived answer to the request is picked up, and others are cancelled.

Please notice to the distributed control of the function. In addition, busses of our system has a function to watch the flow of the information, and the problem caused by the unsuitable location of the specific type manager can be corrected.

6. BASIC CONFIGURATION OF THE OPERATING SYSTEM

6.1 Modular Configuration

Basically, the modular configuration of the operating system (OS) is preferable to fit to the multiprocessor environment, because of it is convenient for the distribution of needed parts of OS to plural processors. The efficient parallel processing, striking off of the redundant part of OS will be prepared easily by the configuration.

However, the decrease of communication between modules and the increase of parallelism between distributed tasks must be considered. In this viewpoint, we devided the OS in modules as to have a logical sizable function.

The fact that the OS of ADMS is a set of type managers, as mentioned in previous section, will support the modularization. Each module is a server which provides a specific function in the OS, and in same time it is a type manager for the object. Thus, the OS in our system is modularized in an unit of type manager, each module is executed as a process, and the communication between modules, and between user and OS, are executed both as the communication between processes.

The last item brings the unification of communication mechanism which are separated in case by case (for example, OS/OS, OS/user, user/user) in the traditional system.

6.2 Communication Mechanism

The communication must be executed irrelevant to the location of target process. That will bring the independency of the software from the hardware, make easy to increase or decrease the number of the server process, and have the improved availability of the system in case of trouble.

The communication must be protected. We can execute the communication between user and OS as the ordinal inter-process communication. That will have the conveniency on the modularization of OS, the dynamic reconstruction of it, and the wide area communication in the system.

For the design of the mechanism, we apply the concept of "the self-identification of data"|Mye78| on the communication, to apply generic instructions and to keep the independency of data by making an explicit unit for the communication.

In our system, the message and the port are prepared as tools for the communication. The message is independent from the port, and they are realized as objects. By the communi-

cation, messages are transmitted to the port from not only sender but receiver, too. Now, we call them as sending message and receiving message.

In case of one way communication, sending message brings data to the port, and data is copied to the receiving message on the port. If the response is needed, the sending and the receiving message are transfered both to the receiver from the port, and sending message brings back the result of the needed operation in the receiver to the sender.

By the mechanism, it is easy to receive data from the port which exists on other processors. Messages can be reused, and each message has a fixed port address. The fact means that the address of a specific message can be decided or changed by the parent process or the system. If we put the processor number in the address, the information can be utilized for the routing of the communication.

The details of the communication method can be stored in the message itself, and we can execute the communication in a generic mode.

Each process has plural messages corresponded to the ports to which said process may communicate. The side effect of this mechanism is the limitation on the number of message, if needed.

As above mentioned, we use a port object and appoint it as a communication media for the communication. The mechanism of appointing of the communication media is used in pipe of UNIX*|Rit74|, Port of Hydra|Wul74|, and Mailbox of StarOS |Jon79|, etc.. By the mechanism following functions can be

---

*UNIX is a trademark of the Bell Laboratories.

realized.

* A multi-server mechanism can be carried out.
* The making of libraries from processes is promoted.
* It is effective for the abstraction of the interrupt and I/O process.
* There are few problems by the communication to the not yet created process.

The destroy procedure of the port is same to other objects.

6.3 Mode of the Communication

The conditional send/receive is applicable. The mode signal is put in the message itself.

The nonblocking send/receive, the multiple wait communication are available, too. The heading of the message has the address to the port, so the procedure to send messages to the remote port is approximately same to the case of local port.

6.4 Process Management

In ADMS, processes under management have one of following status; running, ready, wait, wait-suspended (a status in which a waiting-process is put out from dispatching by the suspended-process), suspended (a status in which a ready-process is put out from dispatching by the suspended-process), and dormant.

And for the transition of the status, following operations are prepared; create-process, start-process, delete-process, exit-process, abort-process, terminate-process, suspend-process, resume-process, sleep-process, and wake-up-process.

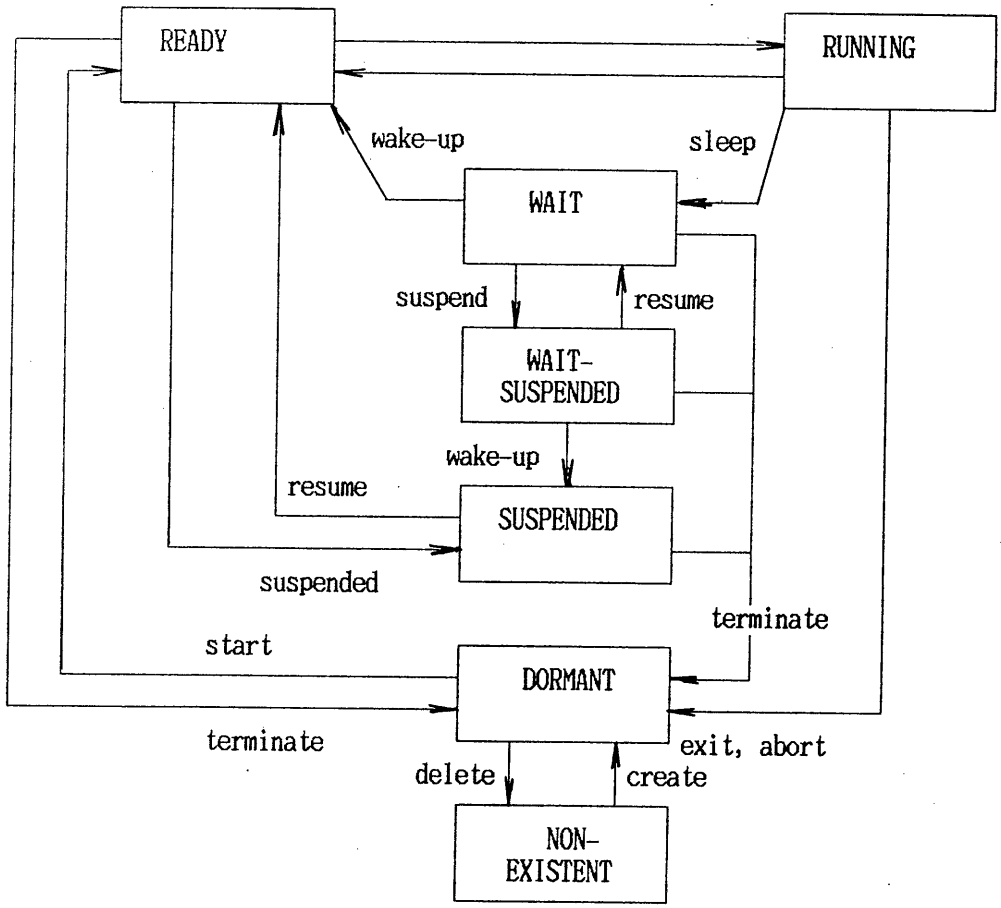

Fig.8 The Status Transition of the Process.

The relation of these items are shown in Fig.8.

On the scheduling of the process, 8 priority levels are prepared and the queue is composed in each level. Then the length of time slot is decided as:

$$T=T_0 \times 2^P$$

here; T : the length of the time slot.

T_0 : unit length.

P : priority level number (0~7).

Still more, the priority level is limited in some ranges by the character of each process, the priority of a process which is brought to the "wait" is changed to one level upper, and the idle process level is introduced as the lowest priority level to keep the processor in ready status.

The operations prepared for the scheduling are; schedule-process, dispatch-process, change-priority, change-ready-queue, set-process-status, disable-interrupt, and enable-interrupt.

The process control block (PCB) is prepared and the important information on processes are stored in it. The PCB is devided in two parts as D-PCB and C-PCB. D-PCB has information of data, and C-PCB has information of capabilities for D-PCB or context. The substance of the C-PCB is the C segment. We have no special fearure on the interrupt operation, and the timer management.

7. CONCLUSION

In htis paper, we have discussed on a loosely coupled multiprocessor system to use for the distributed processing system, especially on the idea concerning to the logical configuration and the operating system.

The implementation of the system is not yet completed now, however, we have no interference with the investigation in present. We are under designing of language processors and the hardware for the system, and we will be able to report on them in near future. Then, we will describe on the result of the evaluation of our works.

ACKNOWLEDGEMENT

We thank to Mr. Takeo WAKAMOTO, who has designed on the process management mechanism and proposed many effective ideas to us. And we thank to Mr. Norio OHASHI, Mr. Kazuhiro WATANABE, and Mr. Sadaji ASANO as exellent cooperators of our works.

REFERENCES

Ama86: S.Amada, M.Tsuchida, Y.Sato: A Loosely Coupled Multiprocessor system: ADMS--Basic Design--, The Tech. Report of Institute of Information Sciences and Electronics, Univ. of Tsukuba, ISE-TR-86-56, pp.1-16, (June 1986).

Fab74: R.S.Fably: Capability-Based Addressing, Comm. ACM, 17, 4, pp.403-412, (Dec. 1974).

Hou81: M.E.Houdek, et al.: IBM System/38 Support for Capability-Based Addressing, The 8th Symp. on Comp. Architecture, pp.341-348, (1981).

Ish84: Y.Ishikawa, M.Tokoro: The Design of an Object Oriented Architecture, The 11th Annual International Symp. on Comp. Architecture, pp.178-187, (1984).

Jon79: A.K.Jones, et al.: StarOS, a Multiprocessor Operating System for the Support of Task Forces, The 7th Symp. on Operating System Principles, SIGOPS, pp.117-127, (1979).

Lin76: T.A.Linden: Operating System Structures to Support Security and Reliable Software, Computing Surveys, 8, 4, pp.409-445, (Dec. 1976).

Mye78: G.J.Myers: Advances in Computer Architecture, John Wiley & Sons, (1978).

Mye80: G.J.Myers, B.R.S.Buckingham: A Hardware Implementation of Capability-Based Addressing, Comp. Architecture News, 8, 6, pp.12-23, (Oct. 1980).

Rit74: D.M.Ritchie, K.Thompson: The UNIX Time-Sharing System, Comm. ACM, 17, 7, pp.365-375, (July 1974).

Tyn81: P.Tyner: iAPX432 General Data Processor Architecture Reference Manual, Intel Corporation, (1981).

Wul74: W.Wulf, et al.: HYDRA: The Kernel of a Multiprocessor Operating System, Comm. ACM, 17, 6, pp.337-345, (June 1974).

Wul81: W.Wulf, R.Levin, S.P.Harbison: HYDRA/C.mmp: An Experimental Computer System, McGraw-Hill, (1981).

INSTITUTE OF INFORMATION SCIENCES AND ELECTRONICS
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI 305 JAPAN

REPORT DOCUMENTATION PAGE	REPORT NUMBER ISE-TR-86-59

TITLE

A Loosely Coupled Multiprocessor System : ADMS

---Logical Configuration & Operating System---

AUTHOR(S)

Sanae Amada

Yutaka Sato

Syu-ichi Suzuki

REPORT DATE	NUMBER OF PAGES
October 16, 1986	27
MAIN CATEGORY Multiprocessor System	CR CATEGORIES C.1.2, C.1.3, D.1.3, D.4.1.

KEY WORDS

Distributed Processing, Multiprocessor, High-Level Architecture, Capability-Based Addressing, Object Oriented, Operating System.

ABSTRACT

We have described on the basic design of ADMS, already |Ama86|. In this paper, we add some results of our works concerning with the logical configuration and the operating system of ADMS to the previous report. The reasons of use of the capability-based addressing, the addressing mechanism, the executive environment of the program, the object oriented environment, and the basic configuration of the operating system are discussed.

SUPPLEMENTARY NOTES