SECOND HUNGARIAN COMPUTER SCIENCE CONFERENCE



# PREPRINTS

## I.

## 2. PROCESS MANAGEMENT

The process is the logical unit of the system. There are two type of processes according to the logical level at which they are active :

- Locally active processes whose activity is not known outside the entity on which they are running.

Depending on the particular application, communications mechanisms may vary: explicit message exchanges or data sharing through a main memory for example.

Process management strategies at this level are local to the single entities and do not influence global process management.

- Processes which are active at the system level sharing global resources. The most natural way of interaction between remote processes is by an explicit message exchange. This mechanism is the most similar to the physical protocols which are generally used for data transmission between remote entities. Message exchange is made by means of two primitives: SEND and RECEIVE.

When calling a SEND primitive a process must specify a pointer to the message and the destination process name. The destination process may have access to the information by calling a RECEIVE primitive specifying the sending process name. The information exchange may be made asynchronous by inserting a shared-buffer in the data path connecting the two processes. Process connection strongly depends on the system modularity level. In a strongly monolithic system in which all linking between variables is made at compile - time, the programmer refers directly to the names of processes as connections between them are all known at programming time.

System reconfiguration requires complete recompilation of the whole system program. A method to achieve a greater degree of flexibility and easier modifiabilityis to decompose the system program into functional modules with standard interfaces. Modules may be compiled separately and a family of system may be generated by various



combinations of modules which are the linked together at system configuration time. In such a system, information about process connection is not available at programming time. A method to achieve this level of modularity is the "port - approach", in which each process defines a set of input and output points (ports) through which messages may be sent and received. Port names are variables local to a single module and are the only interfaces with other modules. Process connection is obtained by mapping input ports with output ports at system configuration time. If the system is distributed among independent entities, information for remote process connection is not statistically available.. Port mapping must be made during execution. References to external processes must still be made through local names and a dynamic mechanism to connect processes is required. In order to avoid misunderstanding at the system level, processes are identified by a unique logical name. A unique mechanism to generate global names must be implemented on each entity.

## 3. RESOURCE MANAGEMENT

In this paper we anlyze the minimum functions that each entity of the system must implement in order to obtain global resource management which achieve the following goals:

- a system with a high degree of reliability and flexibility and with automatic reconfiguration in the presence of local charges;
- a uniform sharing of computing loads amoung the system components;
- avoidance of dead-lock situations.

Physical and logical system resources are of various types and have different accessing modes. In order to obtain a uniform system structuring, access to resources must be made at an abstract level, thus implementing protection mechanisms and hiding some implementation dependent characteristics which are not essential to the functions required.

- RESOURCE MANAGEMENT IN A PROCESS ORIENTED DISTRIBUTED OPERATING SYSTEM.

Sergio COPELLI *
Italy

Norma LIJTMAER **
Italy

## 1. INTRODUCTION

A distributed system is the interconnection of a set of computing modules on which the simultaneous execution of different activities is possible.

Distributed architectures may be classified according to the degree of interconnection of the different entities: these range from a multiprocessor system in which the system components are tightly connected and information exchange is made through a shared main memory, to a computer network in which interactions are less frequent and the components communicate via messages over data - links.

Functions may be distributed amoung several entities, so that all these entities are similar and each one is capable of performing the whole function. If a function is distributed and each entity has access to all data (data structures and procedures) the system is said to have global knowledge. In the system which we analyze, on the other hand, each entity has access to its own set of data and the sets of data manipulated by other entities



for the same function are not directly available.

Sharing of data implies a direct interaction amoung entities by an explicit message exchange. Each entity has the resposibility for the allocation of its own resources and the scheduling of local processes.

The main goal of this paper is to propose a uniform treatment of these two aspects in order to achieve complete transparency at the user level.

dynamically vary according to the system configuration. So the interprocess communication mechanism must rely on logical names which are independent from the physical allocation. Hence there must be a single mechanism in the system to generate unique logical names for processes. Mapping of logical names into physical names is done by the interconnection subsystem, which is not analyzed in this paper.

## 5. COMMUNICATION LEVELS

There are different levels of communication among system processes:

-Communications among user processes in order to have data and program migration among entities. If the information flow goes from a single process to another single process, messages are sent by means of unique logical names. Message broadcasting may also be realized by type-names. In this case a unique message structure is delivered to a set of processes of the same type.

-Communication among user processes and local system processes to access system resources. Different modes may be used: unique logical names or type-names. Another possibility is when a user process wants to communicate with a process of a specific type whose unique name is unknown. The operating system chooses a process among the processes of that type and attaches the name to the requesting process. This allows, for example, uniform load-sharing among many copies of the same process-type.

-Communications among system processes. They allow local operating systems to exchange information concerning the global state of the system in order to optimize the use of the system resources.

SEND and RECEIVE primitives for message-exchange are implemented by the local nucleus. Each nucleus has a table containing the names



of locally active processes. When a process executes a SEND primitive the nucleus enqueues the message to the receiver, if the receiver is locally active. Otherwise, the nucleus sends the message to the communication subsystem which connects the system entities. The same is true for the RECEIVE primitive.

## 6. CASE STUDY

As an example, the process short-term scheduling in a local network of minicomputers with a ring topology is analized. [3]

The goals to be achieved in this project are:

i)Obtaining a completely distributed system;

ii)Avoiding constraints to the modularity of software systems running on network's hosts;

iii)Obtaining a system with a high degree of modularity.

In order to achieve these objectives a Ring Topology has been chosen to interconnect the mini systems.

A ring structure is obtained by connecting the output of system N to the input of system N+1 and the output of the last system to the input of the first system.

The general configuration of the network proposed is illustrated in figure 1.

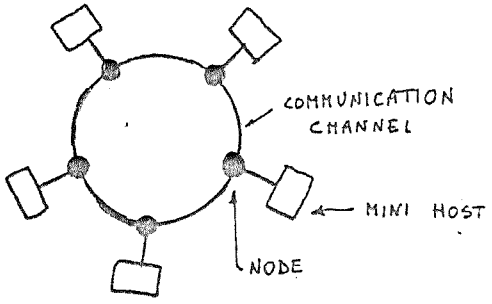


FIG. 1 Ring Configuration.

### 3.1. DESIGN SPECIFICATION OF ABSTRACT RESOURCES

There are many recent programming languages which define programming tools for abstract resorce management. This vary from the "abstract data type" concept, in which a date structure is defined in terms of the operations that can be made on it, to the "monitor" or "path-expressions" concepts [9] , defining the syncronization rules for access to a shared resource.

A monitor variable is defined by a shared data-type declaration and by specifying access procedures. Particular implementations of various instances of a monitor type are automatically made by the abstract machine on which the programming language is defined. The only accessing mode to a monitor - controlled resource is by explicit monitor calls specifying the type of operation to be made on the resource. This process interaction mechanism allows compile - time checking on correct access to resources. This structuring techniques implies that compilation of the whole system must be unique. If the system is decomposed into functional modules by keeping a library of the interfaces defined by the modules check functions may be done at configuration time. In both cases this interaction mechanism is typical of a centralized system in which information about interfaces is statically available.

If the system is distributed in different entities it is only possible to check the interface speicfications at compile time. Process interactions checking must be done at execution time by associating a process to each shared resource.

A resource request is made by sending a message to the process which manages it. This allows for uniform treatment of processes and resources. Static checking is made on the correct use of the message exchange mechanism while correct use of resources is dinamically checked by the associated process.

## 4. SYSTEM STRUCTURE

With the structuring technique previously defined, the global system at a certain level of abstraction may be seen as a set of concurrent processes.

The system components can be different and perform different activities, so local strategies for process management may also be different. In order to easily change local strategies without changing the interfaces with the other entities, the strategies must be logically separated from the mechanisms wich are used to implement them. So each entity has, at the lowest logical level, a nucleus implementing mechanisms which are hardware extensions and which form the environment for process activity. Some of these mechanisms, like the message exchange primitives, form the interfaces between the entities. They are, therefore, identical in the whole system and must be implemented on each entity.

Processes are active at the upper levels and may form different operating systems families at a local level. They can also migrate among the system components but this does not influence the global system.

As the system components may be etherogeneous systems with different physical characteristics, the logical communication protocol is implemented at the process level. At this level it is simpler to change strategies according to specific applications.

As there is no centralized point of control, global information is distributed among the different entities. Local operating systems have direct access to local information and communicate with remote operating systems to share global information. In particular, each entity has a complete knowledge of local processes, but does not know the physical allocation of remote resources and the processors where processes are active. In fact this information may

The mini-systems (the "hosts" of the network) are connected to the communication channel (a unidirectional line) by means of a hardware interface called "node".
No consideration is done about the special characteristics of the host. In fact, hosts may be heterogeneous or homogeneous. Message passing technique is assumed to be the natural way to logically communicate among different hosts.

Each host has a resident nucleus which performs both process management and creation functions.
Process is the logical unit of the system. All resources available in the network are associated to processes. A resource request is done by referring to the associated process.
Processes are referred to by their names. Names contain some type information. Name-creation criteria must be unique in the system.
An example of name encoding is that shown in figure 2.


FIG. 2 Process Name Encoding.

It consists of two parts:
i) Field "type": the type of the resource;
ii) Field "name": the particular name of the resource inside that type set.
As an example, FORTRAN and ALGOL compilers belong to the same type "compiler" with different names.
Each resident nucleus has a list of logical names of processes which are active on each host. Logical names are local variables.
Message exchange is performed by means of two primitives: SEND and RECEIVE.

By means of a SEND function a process $P_s$ sends a message from its output connections (ports) to the input connections of the destination process $P_d$ through a mailbox.
Therefore, all logical and physical operations of putting and getting messages from the ring are transparent at the nucleus level.
Note that $P_s$ doesn't know anything about process $P_d$, it only knows the local names of its own ports.
The communication paths among processes and the mailboxes are fixed by the system nucleus using information about the structure of each software system.
Some of the processes connected are locally resident, others are remote. A simple strategy may be given in order to define local and remote processes. In any case the allocation process is transparent to the user.

Remote processes and remote mailboxes are implemented as a set of virtual processes and virtual mailboxes by an Interface Processor at a lower level. While virtual processes are the logical representation of remote process descriptors, virtual mailboxes are extensions of remote mailboxes. Semaphores ensuring the correctness of the message exchanging mechanism in each virtual mailbox are defined and managed at this level.

The functions of the Interface Processor are:
i) Taking messages from the output mailboxes of local processes connected to remote processes and after attaching to them some information to identify the destination process, putting them in an empty input buffer of the communication subsystem in order to be sent on the ring;
ii)Getting messages from the output buffer of the communication subsystem and according to the information about the destination process, dispatching them in the proper input mailboxes.

The information about the destination process contains the name of the process and a process input port name. The Interface Processor reads information belonging to the nucleus link table. The entries of this table contain information of the type: "Port $p_j$ of process $P_x$ is connected to port $p_k$ of process $P_y$". Although process logical names are unique in the system, this is not enough to ensure a correct processing of messages because a process is generally structured with multiple input and output ports.

At the upper level users are unaware of where physical and logical resources are allocated in the system. This uniform treatment of processes and resources ensures a complete transparency at this level.



## 7. REFERENCES.


[1] Farber, D.J. et al. The Distributed Computing System, 7th Annual IEEE computer Soc. International Conference, Feb. 73.

[2] Neumann, P.J., System Design for Computer Networks, in: Abramson and Kuo (eds.), Computer Communication Networks, Prentice Hall, 73

[3] Casaglia, G.F., Copelli,S. , Lijtmaer,N. Distributed Control For Local Network of Minis: A Design Approach using Microprocessors, 2nd Symposium on Micro Architecture, EUROMICRO 76, Oct.76, Venice.

[4] Walden, D.C., A System for Interprocess Communication in a Resource Sharing Computer Network, Comm. ACM vol.15 n.4 Apr. 72.

[5] Balzer,R.M., Ports-A Method for Dynamic Interprogram Communication and Job Control, Proc. of the Spring Joint Computer Conference vol. 38, 1971.

[6] Hoare,C.A.R., Monitors: An Operating System Structuring Concept, Comm. ACM 17,10, Ott.74, pp.549-557.

[7] Flon,L., Program Design With Abstract Data Types, Department of Computer Science, Carnagie-Mellon Universitt, Pittsburg, Pa., June 75.

[8] Brinch Hansen,P., A Programming Methodology for Operating System Design, Proc IFIP 74 congress, Aug.74, pp.394-397.

[9] Ancilotti,P., Lijtmaer,N.,Boari,M., Metodi per la Specifica del Coordinamento dei Processi Concorrenti, Giornate di studio su: "Programmazione Strutturata: Esperienze e Orientamenti" Milan, June 76.



* Ing. C. Olivetti S.P.A., R&D, Ivrea, Italy and
Scuola Superiore di Studi Universitari e di Perfezionamento, Pisa, Italy.

** Istituto di Elaborazione dell'Informazione del CNR, Pisa, Italy.


This work was partially supported under a research agreement between Olivetti S.P.A., Ivrea, Italy and National Research Counsil, Italy.

*********