CS/TR-92-10

CHORUS/MiX, a Distributed UNIX, on Multicomputers

Bénédicte Herrmann *(LIB, Université de Franche-Comté - ONERA)*
Laurent Philippe *(LIB, Université de Franche-Comté - Chorus systèmes)*
Chorus systèmes, 6, avenue Gustave Eiffel,
F–78182, Saint-Quentin-en-Yvelines (France). Tel: +33 1 30 64 82 00,
E-mail: lau@chorus.fr

Abstract: Currently available multicomputers are generally featured with simple operating system kernels offering mostly communication primitives. The CHORUS technology has been designed for building "new generations" of open, distributed, scalable operating systems. It is based on a small kernel onto which operating systems are built as sets of distributed, cooperating servers. This paper presents a description of CHORUS/MiX on multicomputer, a first step in implementing a UNIX interface suited for this architecture. A discussion of the port on an experimental platform is given. Finally, we describe an IMS T9000-based version of CHORUS and explain how T9000-based multicomputers will benefit from the experiences of the port on the experimental platform.

Keywords: Distribution, UNIX, multicomputers, CHORUS, Transputer

1. Introduction

Multiprocessor architectures tend to be the current answer for the design of high performance computers – supercomputers and mainframes. Instead of designing costly special purpose processors, standard microprocessors are used, the new development costs being concentrated on the interconnection system. Two main architectures are being experimented: shared-memory multiprocessors (SMP) and loosely-coupled multiprocessors, called multicomputers in this paper.

In order to allow the widest and easiest use of these machines it is necessary to adapt standard operating system kernels (eg. UNIX®) to them. Most of the operating system kernels have been

In: Proceedings of Transputer'92, Arc et Senans, France, May 20-22, 1992

® UNIX is a registered trademark of UNIX System Laboratories, Inc. in the U.S.A. and other countries.

adapted to the support of SMP's. A single, parallelized instance of the operating system kernel transparently balances the users load among the processors. The user is generally not aware of the multiprocessor nature of the machine: the parallelized kernel offers a single system image. For those users who need to master the parallelism, for example scientific computing applications, new scheduling techniques (eg. gang scheduling[Black90]) have been designed.

Unfortunately, these systems currently suffer from scalability problems. The I/O and memory buses become the bottlenecks of the system, which must be balanced by costly memory (caching) architectures. From the software point of view, the ability to support a large number of processors implies a very high degree of kernel parallelization, which is hard and costly to achieve. Current systems are generally limited to a few ten's of processors.

On the other hand, multicomputers offer much more promising scalability properties. Based on high performance networks, they use network topologies (mesh, hypercubes, etc) such that additional processors always come with additional network bandwith. In addition the I/O load may be shared by independent nodes. These architectures promise an interesting price/performance ratio. Some currently available systems include thousands of processors. On such multicomputer architectures, each node runs an instance of the operating system kernel. These differents instances communicate via message passing.

Currently available multicomputers are generally featured with very simple operating system kernels (ex: NX[Pier88] on iPSC/2, Helios[Garn87] on transputers) offering mostly communications primitives. Services such as user interaction and device access are provided via separate "host" computers. The multicomputer is generally used as a high performance co-processor connected to workstations and mainly for scientific computing. In order to extend the use of these promising architectures – for example to transactionnal systems – standard operating systems must be adapted. A fast approach is to adapt currently available network operating systems, providing distributed services such as distributed file systems (eg. NFS) and remote execution facilities (eg. rsh). However, we believe that this is not sufficient. As well as SMP's, multicomputers must provide a single system image to their users. In addition, the operating system must be flexible enough to be adapted to the different natures of the nodes of the multicomputer (compute nodes, I/O nodes).
We believe that the CHORUS® microkernel technology is a strong basis for achieving these goals.

In section two we define more precisely the characteristics of multicomputers as well as the main requirements for their operating systems. In section three, we briefly outline the CHORUS micro kernel technology and its application to the construction of a modular UNIX system: CHORUS/MiX. In section four, we discuss the port of CHORUS/MiX on a multicomputer plateform – the iPSC/2. Finally, we describe some of the issues involved in porting CHORUS/MiX on the T9000[Inmos91].

2. Distributed memory multiprocessor

We are interested in distributed memory multiprocessor machines because this type of architecture

® CHORUS is a registered trademark of Chorus systèmes.

offers more scalability and fault-tolerance than tightly coupled multiprocessors. Indeed each compute unit (ie: the processor and memory) is independent and shares less hardware resources with others separate units.

2.1 Definitions related to multicomputers

We will call a *multicomputer*, a totally distributed memory multiprocessor machine. We define a *node* as a set of tightly coupled processors and their supporting environment. So, a multicomputer is a set of nodes interconnected via a uniform scalable network.

We will call the interconnection network of the multicomputers an *internal network* (intnet) and the network used to connect the multicomputer to other machines (eg: Ethernet for workstations) an *external network* (extnet).

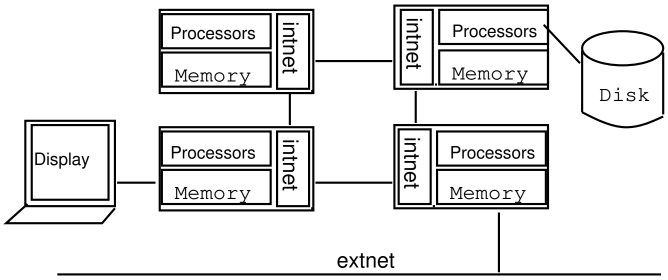

Figure 1. – Multicomputer architecture

In a multicomputer, we define two models of nodes. A *basic node* includes the processors with their memory and a connection to the internal network. A *specialized node* consists of a basic node plus some devices. In multicomputers, each node is independent in the sense that it has its private memory, its own interrupt control, etc., which cannot be handled or accessed by a remote processor. Therefore all the requests to a device have to be handled on the specialized node responsible for that device and nowhere else. The processors may only exchange data by messages.

Examples of multicomputers are:

- the iPSC/2, an hypercube architecture marketed commercially by Intel. Each node is based on a i386 processor and a communication hardware. It scales from 4 to 128 nodes.
- transputer based machines like the T-node[Telmat89] from Telmat. Each node is based on a T800 from INMOS and uses its communication channels to exchange messages. The nodes are interconnected through the C100 routing module, providing a reconfigurable network. T-nodes are basically 16 nodes multicomputers.
- nCube machines: the nCUBE 2[nCUBE90] is an hypercube scaling from 8 to 8096 processors. The nodes are based on a home-made processor and communicate via a home-made communication hardware.

2.2 Operating systems requirements

Currently multicomputers are mainly used for specialized applications (for example mathematical simulation). We can compare the actual use of the multicomputers to batch execution on earlier computer: users must develop their code in a workstation environment (because of the ease of use), then, in order to execute it, they must statically allocate a sub-set of the nodes and download binary programs to the multicomputer. This sub-set cannot be shared with other users, in effect these machines are not real multi-users machines.
Further, applications developed on multicomputers manage their own distribution, placing each task on each node. These applications are architecture dependent. For instance, if an application runs on sixteen nodes it will not benefit from a thirty two nodes machine.
Generally, there are only a few facilities on the multicomputer to manage the needs of general purpose applications such as input and output. It is very important for the user to find on the multicomputer an interface as close as possible to his workstation interface because the user is familiar with this interface and will use it easily. We believe that multicomputers may be interesting as general purpose machines. Yet, users may not accept these new machines if they do not find equivalent facilities or tools as on their workstations. Users want to gain in compute power without losing in programming time, energy and comfort. These requirements must be taken into account by the operating system developers because they are crucial for effective use of multicomputers. Indeed the operating system is the base on which the environment for programming and debugging applications will be developed.

To satisfy these requirements we need an operating system which integrates distribution and provides a standard interface. On these topics we distinguish three main levels of evolution for the operating system functionality:

- The first level of evolution is to provide the same kind of interface on a multicomputer as on a workstation: provide a uniform, standard operating system interface to support a large number of existing tools (eg: databases, graphics interfaces), manage the devices attached to the specialized nodes and provide access to remote resources (files, terminalsetc). In fact, the multicomputer may be seen, from an operating system point of view, as a set of workstations interconnected via a network. For instance this is the level of functionality provided by UNIX on a set of interconnected workstations. This level does not provide transparent access to resources, so the user must master all the distribution of the resources of its application which is, in this case, location dependent. On the other hand the current UNIX technology does not provide a modular implementation so all the kernel must be loaded on all the nodes even if all the code will not be used (for example device management on basic nodes).

- The second level of evolution extends these interfaces to deal with distribution and to help the user to benefit from the underlaying architecture without too much difficulty. It must provide:
  - transparent access to the resources (ie: devices, memory, etc) available on one node by the other nodes. Actually, the access to the processor is not transparent and the user knows which process executes on which node.
  - parallel execution facilities. For instance simple communication facilities (like send and receive) or remote execution.
  - new tools which will help for the development and debug of distributed applications. New debuggers are needed because distributed applications cannot be debugged with standard tools.

  Further the architecture of the operating system must be modular to adapt to the configuration of each node. This level of evolution allows users to write location independent applications. It has been reached by some operating systems as AMOEBA[Tanen90], CHORUS[Rozi88]. or MACH[Acce86]. Yet the distribution of the application still has to be mastered by the user.

- The third level of evolution is to adapt the operating system to the architecture. We believe that the operating system must allow users to develop applications without taking into account the architecture of the multiprocessor. Thus they can develop architecture independent applications. To achieve that, the interface must provide a single system image of the multicomputer. Moreover the operating system must be configurable: on each node we just need the part of the operating system which manages the local resources. We must also integrate new possibilities such as fault tolerance because the probability of a failure is greater with hundreds or thousands of nodes than in a workstation.

UNIX is one of the most widespread operating system interface used on workstations and networks. For this reason it may be a good choice as operating system interface. Unfortunately, the UNIX technology is not designed to easily support the multicomputer architecture. Its primary goal was to manage centralized architectures, and distribution was not integrated in the basic concepts of UNIX. The CHORUS/MiX system provides a standard UNIX interface but integrates distribution concepts at a low level.

3. CHORUS/MiX V3.2

The CHORUS technology is designed to support distributed architectures by integrating the communication at the kernel level and by implementing a distributed virtual memory. The actual state of CHORUS corresponds to the second level of evolution on operating system. Moreover CHORUS is well suited to implement the third level of evolution.

3.1 Overall Organization

A CHORUS system is composed of a small-sized *Nucleus* and a number of *System Servers*. Those servers cooperate in the context of *Subsystems* (e.g. UNIX) to provide a coherent set of services and interfaces to their ‘‘users’’ (application programs). Thus CHORUS/MiX[Herr88] is made of the Nucleus and a UNIX subsystem. It provides binary compatibility with UNIX SYSTEM V3.2 (SCO).
The physical support for a CHORUS system is composed of a set of *sites*, interconnected by a communication network (i.e., an external network or internal network). In a multicomputer each node is equivalent to a *site*. There is one CHORUS Nucleus per site.
The CHORUS Nucleus manages, at the lowest level, the local physical computing resources of a ‘‘computer’’: allocation of local processor(s) (controlled by a real-time multi-tasking executive), local memory (managed by a virtual memory manager), external events – interrupts, traps, exceptions – (dispatched by a supervisor). The CHORUS Nucleus also provide global services by implementing basic abstractions.

3.2 The CHORUS Nucleus basic abstractions

The *actor* is the logical unit of distribution and of collection of resources in a CHORUS system. An actor defines a protected address space supporting the execution of one or more *threads* (lightweight processes always tied to only one actor) that share the address space of the actor. Any given actor is tied to a site.
CHORUS offers message-based facilities (referred to as *IPC*) which allow any thread to transparently

communicate and synchronize with any other thread, on any site. The CHORUS IPC permits threads to exchange messages either *asynchronously* or by synchronous *Remote Procedure Call*.

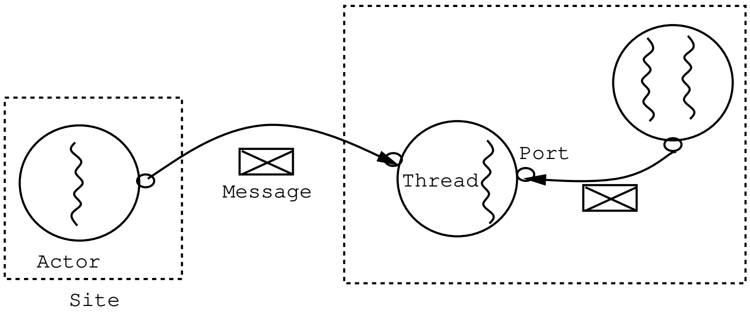

Figure 2. – Basic abstractions

A *message* is an untyped string of bytes composed of a (optional) *body* and an (optional) *annex*. Annex size is fixed (64 bytes currently). Body size is variable. Message passing is tightly coupled with the virtual memory mechanism to enable data transmission without copy. Messages are addressed to intermediate entities called *ports*.

The CHORUS memory management service[Abro89a] provides separate address spaces, associated to actors, called *contexts*. The data of a context is a set of non-overlapping *regions*, which form the valid portions of the context. Regions are mapped (generally) to secondary storage objects, called *segments*. Segments are managed outside of the Nucleus, by external servers called segment *mappers*.

3.3 The UNIX subsystem

UNIX facilities may logically be partitioned into several classes of services according to the different types of resources managed: processes, files, devices, pipes, sockets. The design of the structure of the UNIX Subsystem in CHORUS, called CHORUS/MiX, puts emphasis on a clean definition of the interactions between these different classes of services in order to provide a true modular structure. Thus providing general and network-transparent access to resources such as uniform file and device access.

Each type of system resource (ie. process, file, etc.) is isolated and managed by a dedicated system server. Interactions between these servers are based on the CHORUS IPC.

Several types of servers may be distinguished within a typical UNIX Subsystem: the Process Manager (PM), the File Manager (FM), the Device Managers (DM) and the IPC Manager for the system V IPC. This modular architecture allows the UNIX subsystem to be adaptable, as shown in figure 3.

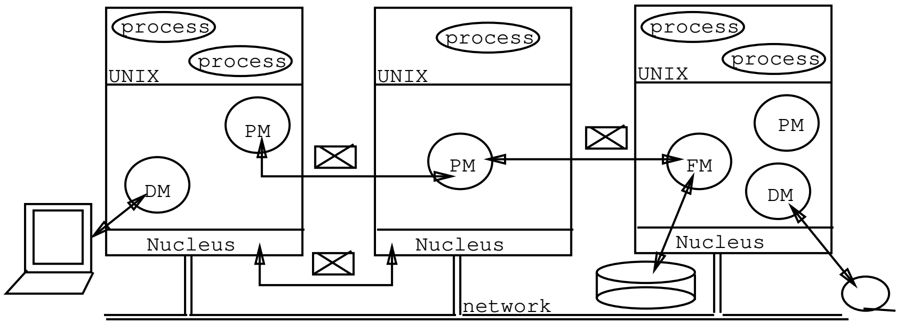

Figure 3. – CHORUS/MiX configuration

3.4 Functional extensions to the UNIX interface

CHORUS provides extensions to the UNIX interface to take benefit of the distributed nature of the system. Such access is not provided by directly invoking the Nucleus but rather through the UNIX Process Manager, in order to eliminate inconsistencies.

- The naming facilities provided by the UNIX file system have been extended to interconnect file systems and provide a global name space.
- The basic extension to process management is to enable remote creation or execution of processes.
- Processes can use the CHORUS IPC mechanisms to communicate transparently over the network.
- Multiprogramming within a UNIX process is possible with multi-threaded UNIX processes.

These extensions provide support for programming parallel applications; in the local case (multi-threaded processes) as in the distributed case (processes communicating by exchange of messages).

The following section describes how CHORUS/MiX V3.2 has been ported on the iPSC/2 multicomputer.

4. CHORUS on multicomputers

The first stage of our project provides the same view of the CHORUS/MiX subsystem on a multicomputer as on a set of interconnected workstations. Our future work will be to develop a distributed UNIX interface suited for multicomputers.

4.1 Mapping of the multi-server model on a multicomputer

There is one CHORUS nucleus on each site, so one nucleus on each node of the multicomputer. On this base we build the UNIX subsystem using the modularity of the system to load only the necessary servers on the appropriate nodes. The Process Manager implements the UNIX interface. There is a PM on each node providing a UNIX interface. However, we do not need to have one PM on each node, for example, performance of requests on files will be greater if the specialized node in charge of the disk does not have to run processes. On the specialized nodes we load the server which manages the local device, for instance a File Manager if the node manages a disk.

However, these considerations are general and must be tuned for each multicomputer.

4.2 An example of multicomputer: the iPSC/2

The iPSC/2[Close88] is a hypercube architecture marketed commercially by Intel Scientific Supercomputer Division. A basic node of the iPSC/2 is composed of a processor (Intel 80386) with 12 Mbytes of RAM (can be expanded to 16 Mbytes) and a DCM (Direct Connect Module) communication module, reachable by the CPU via advanced DMA. This communication module allows connection with up to seven basic nodes and one specialized node. Specialized nodes are basically nodes with an SCSI interface which supports disks, network (Ethernet) or other devices. The hypercube structure is composed only of basic nodes, specialized nodes may be added after.

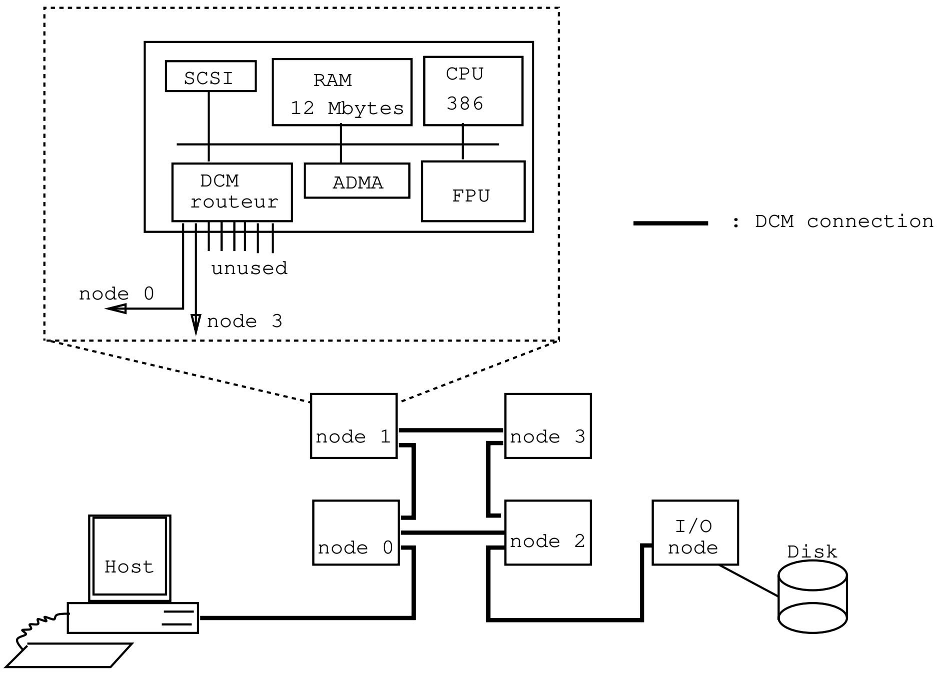

Figure 4. – iPSC/2 Architecture

The nodes can communicate with each other via the DCM. Routing between any two nodes is automatic and does not require CPU intervention. Connected to node 0 via a DCM, there is a host station (PC/AT) which is used to download the programs to the nodes and centralize the information coming from the nodes. This offers a very centralized implementation of the hypercube because all the applications must pass through the host to reach the nodes, after which they are driven from the host.

4.3 CHORUS/MiX on iPSC/2

To get a platform to investigate single system image functionality we have ported CHORUS/MiX on the iPSC/2. In the following sections we concentrate on some of the more important issues that we addressed when carrying out this port.

4.3.1 Nucleus

The traditional CHORUS testbed is a set of COMPAQ's interconnected via a network. So the CHORUS nucleus is already running on Intel 386 based machines. Most of the nucleus code has been reused, we have just changed some machine dependent parts of the nucleus: interrupt management, site number allocation, etc.

4.3.2 Communication and protocols

The communication part of the nucleus is independent of the network. It implements only local communications. When sending a message to a remote site the nucleus locates the destination site and accesses a communication server, called network protocol (NP). After adding the necessary headers to the message, the NP server calls the network driver. We can have several NP on the same site if needed.

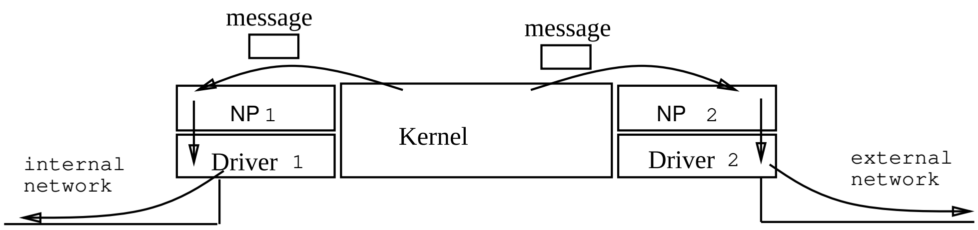

Figure 5. – Message send

We have implemented a network protocol NP1 dedicated to the iPSC/2 network. To optimize the exchange of messages we avoid copying data, using the hardware features and the memory management services of the CHORUS nucleus. In this case the protocol has been implemented in the same actor as the driver, mainly for performance reasons.

From the user point of view the host is a gateway to the hypercube and in particular the connection between the host and node zero. Unfortunately the host does not run CHORUS (we cannot use the CHORUS protocol) and uses the NX protocol (NX is the protocol of the native system) to communicate with node 0. So we have simulated the NX protocol in a network protocol, NP2, which translates NX messages into CHORUS messages. NP2 is implemented on node zero. It provides communication between the external network and the iPSC/2. Actually this is the only way to interact between the hypercube and the "outside" world, in particular to exchange data.

All the communication used at this moment in the cube is actually implemented in the same actor: two protocols and two drivers. Indeed the two protocols share the same hardware structures and interrupt handling thus they cannot be separated. Yet their code and data are mostly distinct so we may have some nodes which communicate with the host, others which do not.

4.3.3 CHORUS/MiX

As noted in section 4.1, when mapping CHORUS/MiX onto each hypercube node, we can take advantage of CHORUS modularity and only use the required server.

4.3.3.1 The Process Manager

The Process Manager (PM) handles all the system calls issued by a process. The PM dispatches the requests to the corresponding servers. For instance on a *read(2)* request, the PM generates a message and sends it to the File manager which will handle the request. Thus, due to the transparency of the IPC, the FM may be located on a remote site.

4.3.3.2 The File Manager

On a multicomputer we need at least one console and one disk for all the nodes to implement the UNIX processes semantics. Yet one is sufficient because all the PMs can access the same File Manager.

Our iPSC/2 configuration has a specialized node with disks. But these disks cannot be used by the standard CHORUS/MiX File Manager because they are not in standard UNIX format. As the purpose of our work is not to develop a file server, we choose to use the disk of the host which is already in a UNIX format. The problem with this disk is that the host does not run CHORUS. So we cannot run the File Manager on the host.
Generally the file management part of the FM accesses the disk management part using blocks: it reads or writes a disk block pointed out by a number. To solve our problem we disassociate[Arm91] the file management (path, inodes, etc) which runs on a node from the disk management (block access) which runs on the host in a UNIX process. The FM and the host process communicate using the NP2 network protocol.
On a disk access, the FM sends the request to a process on the host. The process performs the request on the disk of the host: it reads or writes the designated block. Then it sends the answer to the File Manager. In fact, the host process is used as a disk I/O controler, I/O requests being issued using NP2 messages.

4.3.3.3 Device management

The only usable ttys of the iPSC/2 are the host ttys as we do not have any specialized node with a display.
In CHORUS the Device Manager is mainly used to implement the lines disciplines and display management. In our case the display management is already done on the host by UNIX. So we provide just the basic characters display functions and the rest is done by a process on the host. Single

character device driver has been implemented in the FM which communicate with dedicated processes on the host in order to access the host ttys.

As one can access every ttys, users running X windows on the host can use as many windows as they want. As in standard UNIX an application can display traces in different windows by opening different devices. So different processes can display traces in different windows.

4.3.4 The CHORUS/MiX configuration on the iPSC/2

The first lesson we learned from this implementation is the way to configure the UNIX subsystems on the nodes.

We have explained already how the modularity of the CHORUS/MiX subsystem is used to implement it on multicomputers. On node 0 – which is seen as a specialized node in this case – there are two protocols implemented in the communication manager (intnet protocol plus NX protocol). We choose node 0 because it is physically closer to the host than the other nodes. The File Manager is also running on node 0 for performances reasons because it uses the NX protocol. Thus the File Manager is on the same site as the NX protocol, to access files on the host.

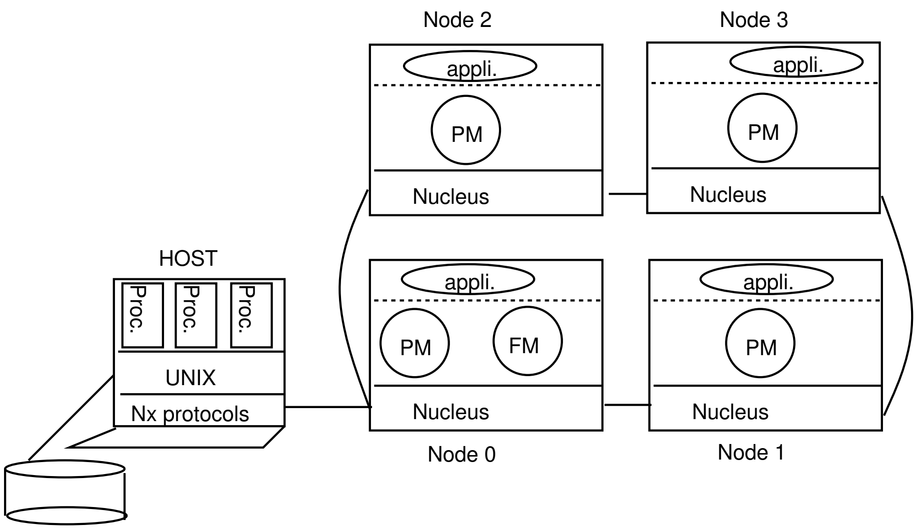

Figure 6. – CHORUS configuration on the iPSC/2

4.3.5 What are the benefits of this UNIX interface?

The interface provides standard UNIX tools on the iPSC/2, for example users can run a shell, compile using *cc* or debug with *gdb*. Moreover these tools are run on CHORUS/MiX without being recompiled, thus benefiting from binary compatibility with UNIX.

Users benefit also from the distribution extensions, both at the shell level and at application

programming level.
At the shell level the user can execute his commands on a remote site using the *remex* command:

remex **-***s*[site number] command

This command keep the environment of the process. For example this command can be used to execute, from the user shell, a new shell on another node. Moreover if the user redirects traces he may have two shells, on two nodes, each using its own window to display traces. This facilitates the debugging of distributed applications.
Users can also use the distributed signal management to remotely control their processes. For instance a process can be killed on a remote node without using remote execution.

At programming level, the extended UNIX interface allows the user to write parallel applications. Each CHORUS/MiX process system context has a default site execution number. This site number can be changed using the *csite(2)* command. On *exec(2)* the PM will use this site number to set the execution site: if the default site number does not correspond to the local site number the PM will perform a remote execution. So we can write parallel applications using *csite(2)*, *fork(2)* and *exec(2)*. The master process which starts the application sets the execution site for its children which will be executed on remote sites. Then all these processes can communicate using the CHORUS IPC facilities.
Moreover this application can be debugged first locally just by setting all the execution site numbers of the children processes to the local node number. Thus we will debug the application locally using a standard debugger. Most of the bugs can be fixed this way, yet not all because the application will run in pseudo parallelism. To distribute the application we will just set the execution sites of the children processes to remote sites.

4.4 CHORUS suited for multicomputers

The CHORUS implementation of UNIX is specially suited to multicomputers because of its modularity and adaptability. It provides a clever management of the multicomputer resources. Moreover it extends the UNIX interface to benefit from the distribution.

This UNIX interface allows the user to execute parallel applications, and is therefore suited to take advantage of a multicomputer architecture. It also provides the same level of functionality as NX and, in the same time, a full UNIX interface.

Applications can be tested on a single machine, and then distributed throughout the network, without any modification necessary to adapt to a new configuration.

We have been using the current CHORUS/MiX V3.2 as a platform to experiment with new mechanisms required to extend the MiX interface to provide a single system image interface.

5. CHORUS on T9000 based multicomputers

In this part we will present the port of the CHORUS nucleus on the T9000[Albin91], first. Actually this port has not been realized since the T9000 is not available but the nucleus run on a T800 based simulator of the T9000. Then we will describe how we will use our experience of multicomputers to port CHORUS on transputers based multicomputers.

5.1 Porting CHORUS onto the T9000

In order to port the CHORUS operating system onto a new hardware architecture the Nucleus must be adapted in order to offer the same set of essential generic services to the higher-level components of the system. These generic services are defined by a clear functional Nucleus interface and by a description of the basic abstractions exported by the Nucleus.

5.1.1 CHORUS threads and T9000 Processes

CHORUS threads are implemented as independent T9000 *processes*. Threads in *supervisor* mode are mapped onto L-processes and run without memory protection or address translation. Each of these L-processes is associated with a T9000 *trap-handler* which is responsible for monitoring the traps and exceptions caused by the thread.

Threads in *user* mode are mapped onto P-processes, and run in a protected address space. Each user thread is controlled by a private *stub-process*, which represents the supervisor mode of the thread.

5.1.2 Scheduling

The CHORUS priority is not to be confused with the two-level priority scheme managed by the T9000 hardware scheduler, hereafter referred to as the *low-* or *high* priority. T9000 processes implementing CHORUS threads are running at low-priority, in order to allow for interrupt preemption. The wider 255-level priority is managed internally by the Nucleus on top of the hardware low-priority.

The context switch mechanism is implemented using a combination of the T9000 hardware scheduler and semaphore atomic operations. For the purpose of scheduling, each thread is associated with a private T9000 semaphore. This scheme has the advantage that there is no need to save and restore any context information since the T9000 hardware has built-in capabilities for managing the scheduler and semaphore hardware queues. Instruction and stack pointers are automatically updated.

5.1.3 Hardware Mechanisms: Traps, Exceptions and Interrupts

Traps and exceptions may occur while the currently running thread is in user or supervisor mode. In the former case, the control is passed to the stub process controlling the thread. In the latter case, the trap handler connected to the thread is responsible for servicing the event. Interrupt sources - timers, event pins and communication links - are monitored by dedicated high-priority processes. These processes, named *vectors*, wait for events to occur on the various sources. When an interrupt source is triggered, the corresponding guardian is awakened and preempts the currently running thread. The vectors and the generic part of the interrupt handlers implement the part of the supervisor not covered by the stub and trap handler processes.

5.1.4 Memory Management

The protected mode offered by the T9000 hardware, with its four per-process independent logical address regions is the base on which the region and context abstractions are implemented. In this model, each thread, if running in protected mode, is assigned four T9000 logical address regions.

The memory management module of the Nucleus must offer a fixed interface to the higher-level parts of the system (region creation and duplication, segment mapping, etc). Because of the T9000 memory architecture characteristics and for efficiency reasons, the full CHORUS memory management module has been re-designed. As an example, it was not possible to implement advanced techniques like copy-on-write or -reference. Regions are loaded or copied entirely at creation time. The buddy system has been adapted for speeding up memory allocation in some cases (like for finding small communication buffers). Relocation algorithms are used to maintain a low level of fragmentation, and are implemented on top of the very efficient block move operations allowed by the T9000.

5.1.5 CHORUS IPC

Remote communications are implemented on top of the T9000 Virtual Channel mechanism. Dedicated vectors act as intermediate agents between the NDM and the actual physical links. This design provides fast response for communication primitives and offers a consistent interface to the portable communication module of the Nucleus, which can consider communication links as normal interrupting devices.

The CHORUS Nucleus IPC module contains a dedicated routing component, the RSM (Routing Site Module), and a reliability component, the FU (Fragmentation Unit), for providing the adequate level of communication semantics to the upper layers.

Routing and location services at the lowest level as well as packet fragmentation are automatically handled by the T9000 Virtual Channel Processor. More complex routing schemes are also managed by the C104 network at the hardware level. The adaptation of the CH IPC on the T9000 takes advantage of the high-reliability and automatic routing capabilities of the hardware.

5.2 Adapting CHORUS/MiX on T9000 based multicomputers

Most of the experiences acquired in the port of CHORUS to the iPSC/2 are reusable on all multicomputer platforms. The main reason is that the Nucleus provides the same interface, regardless of the processor type. There are three main topics on which our experience in porting CHORUS/MiX™ on the iPSC/2 has been useful:

— booting the kernel on the multicomputer. Generaly multicomputers are initialized by one host (ie. node or workstation). This implies that the schemes used to load the operating system on the nodes are equivalent on multicomputers.

— implementing the IPC part of the kernel. Although the communication hardware is not the same on all multicomputers, the structure of the communication servers are the same. It means that the drivers will be differents but the protocols are equivalent (ie. frame).

— adapting the multi-server model of CHORUS/MiX™/MiX to the multicomputer characteristics. The experience acquired in configurating the CHORUS/MiX™/MiX sub-system on each node (ie. starting the needed servers on one node) will be re-used in the implementation of CHORUS/MiX™/MiX on a T9000 based multicomputer.

Moreover, most of the issues encountered during the port to the iPSC/2 will be the same for other architectures. For instance, not all multicomputers have disks and if the host does not run CHORUS, our solution (described in 4.3.3.2) can be used to implement a virtual disk. The same can be use for device and network management.

6. Conclusion

In the actual state of our project we have addressed some of the issues raised by the design of the operating system needed for multicomputers. On that topic, we can conclude that micro kernel technology is well suited for multicomputers; further the CHORUS implementation of UNIX based on the micro kernel model is easily adapted to multicomputers. Actually the current state of the project does not cover all the needs of operating systems dedicated to multicomputers.

The CHORUS/MiX system is further being developed to provide what has been termed *single site semantics* (SSS). This will make it possible to create the illusion of UNIX running on a single processor while taking advantage of the availability of a number of loosely coupled processors. The IMS T9000 Transputer will be one of the first processors on which CHORUS/MiX SSS will be implemented.

7. References

[Abro89a] Vadim Abrossimov, Marc Rozier, and Michel Gien, ‘‘Virtual Memory Management in CHORUS,’’ in *Lecture Notes in Computer Sciences*, Springer-Verlag, Berlin, Germany, (18-19 April 1989), p. 20.

[Acce86] Mike Accetta, Robert Baron, William Bolosky, etc., ‘‘Mach: A New Kernel Foundation for UNIX Development,’’ in *Proc. of USENIX Summer’86 Conference*, Atlanta, GA, (9-13 June 1986), pp. 93-112.

[Albin91] Lawrence Albinson& all., ‘‘Unix on a loosely coupled architecture: the CHORUS/MiX approach,’’ in , (September 91), pp. 321-359. CS/EX-91-49

[Arm91] Francois Armand, ‘‘Offrez un Processus à vos Drivers!,’’ in *Proc. of AFUU’91 Conference*, Paris, France, (91/01), pp. 16. CS/TR-91-8

[Black90] David L. Black, ‘‘Scheduling and Resource Management Techniques for Multiprocessors,’’ in , Carnegie Mellon University, Pittsburgh, PA, (July, 1990), pp. 111. CS/EX-91-78

[Inmos91] Inmos, ‘‘The T9000 transputer products overview manual,’’ in *First Edition*, Inmos limited, (1991).

[Close88] Paul Close, ‘‘The iPSC/2 Node architecture,’’ in *Technical Report*, Intel Scientific Computers, Portland, OR, (88/06), pp. 43-50.

[Garn87] N. H. Garnett, ‘‘HELIOS - An Operating System for the Transputer,’’ in *Proc. of OUG-7, 7th occam User Group Technical Meeting*, Traian Muntean ed., IOS, Grenoble, France, (14-16 September 1987), pp. 411-419.

[Herr88] Frédéric Herrmann, François Armand, Marc Rozier, Michel Gien, etc., ‘‘CHORUS, a New Technology for Building UNIX Systems,’’ in *Proc. of EUUG Autumn’88 Conference*, EUUG, Cascais, Portugal, (3-7 October 1988), pp. 1-18.

[Intel87] Intel, ‘‘iPSC/2 System,’’ in *Product and Market Information*, Intel Scientific Computers, Portland, OR, (87/08), pp. 23.

[nCUBE90] nCUBE, ‘‘NCUBE 26400 Series Computer,’’ in , Technical overview, Beaverton, OR, (1990), pp. 17.

[Pier88] Paul Pierce, ‘‘The NX/2 Operating System,’’ in , Intel Scientific Computers, Portland, OR, (88/06), pp. 51-57. CS/EX-88-417

[Rozi88] Marc Rozier, Vadim Abrossimov, François Armand, Ivan Boule, Michel Gien, etc., ‘‘CHORUS Distributed Operating Systems,’’ *Computing Systems Journal*, vol. 1, no. 4, The Usenix Association, (December 1988), pp. 305-370.

[Tanen90] Andrew S. Tanenbaum, etc., ‘‘Experiences with the Amoeba System Distributed Operating System,’’ *Communications of the ACM*, vol. 33, no. 12, (December 1990), pp. 46-63.

[Telmat89] Telmat informatique, ‘‘T-node overview,’’ in , (juin 1989), pp. 1.

CONTENTS

LIST OF FIGURES