## References


[1] David May and Peter Thompson. *Transputer and Routers: components for concurrent machines*, Tokyo, Japan, 17-18 May 1990. Proceedings of the 3rd Transputer/Occam International Conference.

[2] Clive Dyson. *INMOS H1 architecture revealed*, New Electronics, September 1990, pages 21–24.

[3] Roger Dettmer. *Great Expectations, the H1 Transputer*, IEE Review, December 1990.

[4] Vadim Abrossimov, Marc Rozier, and Marc Shapiro. *Generic Virtual Memory Management for Operating System Kernels*, Proceedings of the 12th ACM Symposium on Operating System Principles, Litchfield Park, AZ, 3-6 December 1989.

[5] François Armand, Michel Gien, Frédéric Herrmann, and Marc Rozier. *Revolution 89, or Distributing UNIX Brings it Back to its Original Virtues*, Proceedings of Workshop on Experiences with Building (and Multiprocessor) Systems, pages 153–174. Ft. Lauderdale, FL, 5-6 October 1989.

[6] François Armand, Frédéric Herrmann, Jim Lipkis, and Marc Rozier. *Multi-threaded Processes in Chorus/MIX*, Proceedings of EUUG Spring' 90 Conference, Munich, Germany, 23-27 April 1990, pages 1–13.

[7] Marc Rozier, Will Neuhauser, and Michel Gien. *Scalability in Distributed Real-Time Operating Systems.* IEEE, Washington, DC, May 1988.

[8] Dick Pountain. *Virtual Channels: The next generation of Transputers*, Byte, April 1990.

[9] Alain Rosset, and Christian Tricot. *The Volvox Machines.* ARCHIPEL S.A. Technical Report, December 1990.

### 5.3.3 *Server Consistency*

Duplication and distribution of system servers raise the issue of server consistency.

First, since objects managed by dedicated servers may be accessed concurrently by remote entities (such an object may typically be a Unix file or device), it is the responsibility of the server managing the object to guarantee that its consistency is maintained. Multi-threaded servers use locking mechanisms and semaphore operations to synchronise the accesses to the underlying objects.

Second, since servers may be duplicated in order to maintain a high degree of availability and to provide support for fault-tolerance, care must be taken to ensure that services are executed with the required semantics (at-most-once, exactly-once, etc). The servers themselves do rely on the underlying IPC addressing semantics to implement these higher level requirements.

## 6 Conclusion and Future Trends

In this paper, we showed how two state-of-the-art technologies - a micro-kernel based operating system architecture and a high performance communication based processor technology - are combined to build multiprocessor systems supporting a standard operating system.

Naturally, it is expected that these technologies continue to impact and enrich each other over the years to come. There is no doubt that distributed systems and their flexibilities are the core of many emerging application requirements; and that the Transputers and the CHORUS operating system are well placed to be at the forefront of these developments.

One particular area for further development is exploitation of the fault-tolerant capability of the Transputer networks in a CHORUS environment. Provisions will be made for access by applications to load balancing, both in terms of computation and communication, when running under CHORUS . These features, coupled with the time-out capabilities of the Transputer communication mechanism, the possibility of multipath message routing and the process migration paradigm open the way for sophisticated fault-tolerant and reconfigurable systems.

The evolution of the Transputer from its early days, where it offered virtual processing, to the upcoming H1 with virtual processing, virtual communication capability, and enhanced memory protection/management is expected to continue. In the future the virtual processing and communication capability of the Transputers will be complemented with full virtual memory thus allowing the more sophisticated memory management models offered by CHORUS operating system to be fully exploited.

## 7 Acknowledgements

We would like to thank Lawrence Albinson, Pascal Piovesan, Julian Wilson, Roger Shepherd, Clive Dyson and Bob Krysiak for their technical contribution to this project.

There is an OM on each site supporting a disk. An OM provides Unix File System management and acts as a *mapper*, acting as a segment server to the memory management module. The OM code is mainly machine independent, except for the low-level device driver part that directly interacts with the disk hardware. Again, this hardware-dependent part of the server uses the service of the Supervisor for connecting interrupt handlers to disk interrupts.

A DM is used on a site whenever tty's ,pseudo-tty's or bitmaps are connected to that site. As for the OM, the porting effort is limited to the actual low-level device driver part of the server.

The SM is the server which manages internet network protocols such as TCP, UDP and IP accessed through the BSD socket interface. The SM does not contain any machine-dependent code, except for a very small part concerned with byte ordering management functions. The actual low-level network driver code is handled by the NDM, described in subsection 5.1.6.

## *5.3 Distribution of CHORUS Servers*

### *5.3.1 Modular Architecture*

The subdivision of the MiX operating system into a set of cooperating servers not only provides functional modularity (each server implements a particular set of services), but also *distribution modularity.*

Distribution modularity refers to the possibility of distributing MiX servers across the sites of the H1 network. As MiX servers use the CHORUS Nucleus IPC mechanisms for communicating together, they implicitly benefit from all the advantages allowed by the flexibility and transparency of these mechanisms.

The different addressing semantics (functional, broadcast) combined with port grouping and migration, offer the basic tools for transparently managing the distribution of the MiX servers.

Server distribution among the sites of the network may dynamically be modified to adapt to the possibly changing network topology, transient traffic or system load variations.

### *5.3.2 Remote Accesses*

As mentioned earlier, there is an OM on each site supporting a disk. Unix processes running on diskless site have access to disk abstractions (file systems, home directory, etc) by transparently communicating with a distant OM. Note that the PM managing the process does not need to know the actual location of the OM on the H1 network, since it relies on the services of the Nucleus transparent IPC mechanism.

Other servers, like the Socket Manager (SM), do not need to be duplicated on all sites. A SM is typically installed at the H1 site responsible for handling the ETHERNET connection, and at strategic locations on the H1 network. Unix process socket communications are serviced locally if the process site has a SM, or by a remote SM accessed transparently by the IPC mechanism.

PMs, which must be present on all "Unix sites", also provide support for remote operations. A Unix process may request for the execution of a child process on a remote site (this is known has the remote fork mechanism). It can also migrated from one site to another. In both cases, the Nucleus IPC mechanisms and the global naming paradigm are used by the PMs to manage the distribution (handling of signals, file contexts, etc).

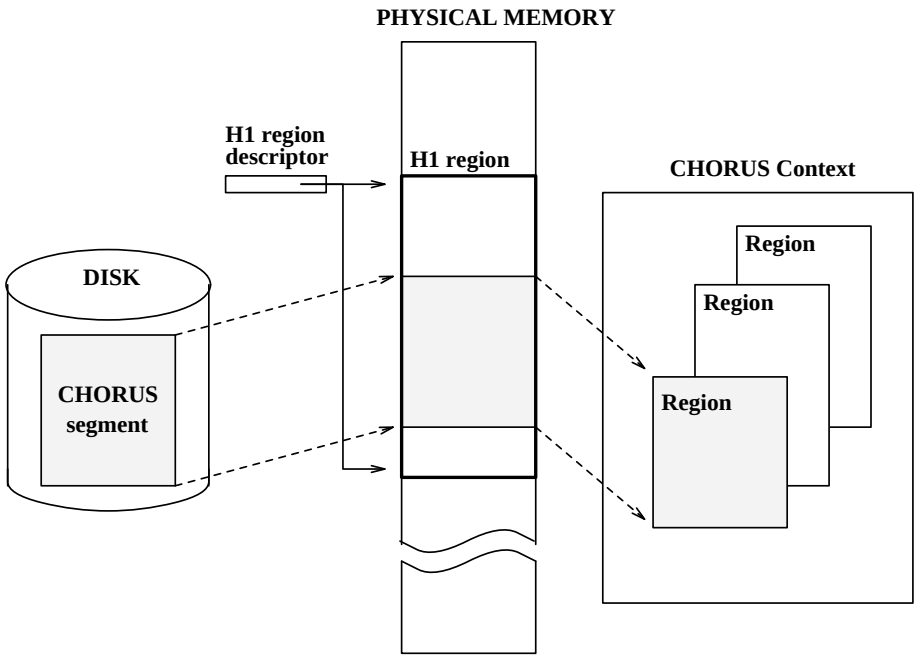


Figure 6: CHORUS segments and regions - H1 regions

communication capabilities. The NDM (Network Device Manager) is the part of the system that manages the communication hardware.

For local communications (i.e. between actors residing on the same site), the Nucleus uses the services of the memory management module for transferring data between the sending and receiving contexts. The transfer is performed using the very fast block move operation offered by the H1 hardware.

Remote communications are implemented on top of the H1 Virtual Channel mechanism. Dedicated guardians (see subsection 5.1.4) act as intermediate agents between the NDM and the actual physical links. This designs 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.

Routing and location services at the lowest level as well as packet fragmentation are automatically handled by the H1 Virtual Channel Processor. Higher level communication primitives (broadcast and CHORUS object location) are handled by the Nucleus.

## *5.2 Porting CHORUS/MiX Servers*

MiX servers run on top of the CHORUS Nucleus. Several types of servers may be distinguished within a typical Unix subsystem: Process Manager (PM), Object Manager (OM), Device Manager (DM) and Socket Manager (SM).

The PM maps Unix processes onto CHORUS abstractions (actor, thread and regions). The PM depends on the hardware characteristics for the management of process contexts and the handling of the low-level Unix trap mechanism. It does so using the services of the Supervisor, which offers routines to connect trap and exception handlers, and to manipulate processes contexts. The PM maps the different memory segments representing a Unix process (text, data and stack segments) on top of distinct CHORUS regions. There is a PM per "Unix site".

occur on the various sources.

When an interrupt source is triggered, the corresponding guardian is awakened and preempts the currently running thread[2]. The guardian saves the current thread's context, updates the interrupt nesting level and starts a process (the actual *interrupt handler*) for servicing the interrupt. The guardians, the context switcher and the generic part of the interrupt handlers implement the part of the Supervisor not covered by the stub and trap handler processes.

The Supervisor is responsible for recording the handlers that the Nucleus or CHORUS actors have connected to hardware interrupts. Interrupt handlers are run at low-priority in order to allow for interrupt nesting and priority.

When the interrupt handler has terminated his job, the preempted thread must be restarted. The H1 hardware characteristics impose that this be done via a high-priority process. Using the saved thread context frame, this high-priority process, called the *context starter*, restores the processor state to the state existing before the interrupt.

Interrupt masking and priority are implemented with a combination of H1 specific instructions (**intdis** and **intenb**) and of H1 semaphores.

### *5.1.5 Memory Management*

The actor is the unit of distribution in the CHORUS system. An actor defines a *protected address space* supporting the execution of one or more threads. The CHORUS memory management service provides separate address spaces (hereafter named *contexts*) associated with each actor, and efficient and versatile mechanisms for data transfer between contexts, and between secondary storage and contexts. A context is represented by a set of non-overlapping regions, and each region is associated with a *segment* by the CHORUS memory management module (segments typically represent some form of backing storage, such as the contents of a file).

The protected mode offered by the H1 hardware, with its four per-process independent logical address regions[3], 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 H1 logical address regions. A H1 region may hold the data of one or more CHORUS regions. If more than one CHORUS region are encapsulated within a H1 region, they must have the same protection and execution privileges.

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 H1 memory architecture characteristics, some memory management techniques, like demand paging and copy-on-write or -reference, cannot be implemented. Regions are therefore loaded or copied entirely at creation time. 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 H1. Figure 6 shows the relationship between CHORUS regions and segments with H1 regions.

### *5.1.6 CHORUS IPC*

CHORUS uses communication as the basic function of the operating system. IPC mechanisms must therefore be as efficient as possible and use at best the hardware

[2]Recall that threads run as low-priority H1 processes and can therefore be preempted at any time by high-priority ones.

[3]We make the distinction between H1 region (hardware unit of memory protection) and CHORUS region (contiguous range of logical addresses within a context).

### *5.1.2 Scheduling*

Synchronous (de)scheduling, or context switch, refers to the deliberate action performed by the currently running thread in order to de-schedule itself and to schedule another thread.

The Nucleus uses the notion of thread *priority* to implement its real-time scheduling policy. The unique criterion for the processor allocation is priority: at any time, the running thread is the ready thread whose priority is the greatest.

This priority-based scheduling technique is also used to implement the time-slicing mechanism. CHORUS priorities are divided into two sets[1]: threads whose priority stands in the upper set obey the strict real-time scheduling strategy, while threads in the lower set are subject to a simple time-slicing strategy.

The CHORUS priority is not to be confused with the two-level priority scheme managed by the H1 hardware scheduler, hereafter referred to as the *low-* or *high-* priority. H1 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 itself has been designed to take advantage of the efficiency of the H1 hardware capabilities. It is implemented using a combination of the H1 hardware scheduler and semaphore atomic operations. This scheme has the advantage that there is no need to save and restore any context information since the H1 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 and Exceptions*

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.

A trap is caused by the execution of the H1 `syscall` instruction. By this mechanism, the thread requests some specific service from the Nucleus. The type of service is passed via the H1 integer evaluation stack, and the thread has pushed the arguments on its stack. The Supervisor maintains trap routines internally and acts as a dispatcher between the hardware trap mechanism and the actual servicing of the request. As mentioned earlier, the Supervisor code is actually implemented by the stub and trap handler processes.

Exceptions are all the events different from traps that may cause the running thread to enter the Nucleus: floating point exception, memory violation, invalid instruction, etc. The action performed by the Supervisor in that case depends on the type of exception. As an example, a memory violation may occur when a user thread's stack pointer goes beyond its memory protection range. In that case, the stub extends the stack region and restarts the thread.

### *5.1.4 Interrupts*

Interrupt sources - timers, event pins and communication links - are monitored by dedicated high-priority processes. These processes, named *guardians*, wait for events to

[1]CHORUS priority values range from 0 to 255. Typically, values from 0 to 127 are reserved for real-time threads, and 128 to 255 for time-sliced threads.

| | |
|---|---|
| Actor | unit of resource allocation, and memory address space |
| Thread | unit of sequential execution |
| Region | unit of structuring an Actor's address space |
| Segment | unit of data encapsulation |

Table 1: CHORUS Nucleus basic abstractions

The Supervisor contains code and data structures that implement the primary hardware event handling (traps, exceptions and interrupts) and the mapping of threads to hardware resources. It also implements thread context switching. Hardware dependencies of the region object are defined and managed by the machine dependent memory management module.

The following sections explain how the CHORUS abstractions are mapped onto the H1 hardware.

### *5.1.1 CHORUS Threads and H1 Processes*

CHORUS threads are implemented as H1 *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 an H1 *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. The stub is responsible for setting up the memory protection regime, for starting the P-process implementing the thread and for handling the trap and exception events caused by the thread.

The code run by a stub or a trap handler as a result of a trap or exception is part of the Nucleus. Other parts of the Nucleus are contained in the guardian and the context starter modules, described in subsection 5.1.4.

Figure 5 shows the mapping between CHORUS threads (user and supervisor) and H1 processes.

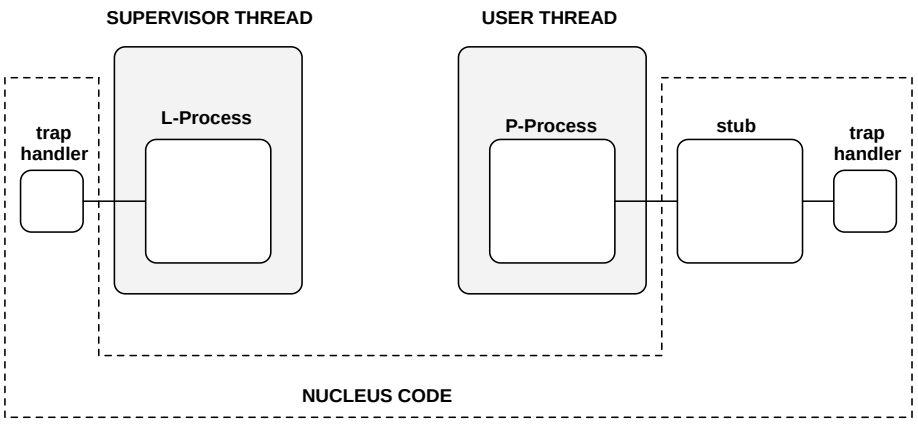


Figure 5: CHORUS threads - H1 processes

group facilities provides a sound basis for doing dynamic reconfiguration and developing fault-tolerant applications.

#### *4.2.4 Multi-threaded Unix Processes*

Multiprogramming within a Unix process is possible with a Unix thread (*u_thread*). A *u_thread* can be considered as a lightweight process within a standard Unix process. It shares all the process resources and in particular its memory space and opened files. When a process terminates by exit, all the *u_threads* of that process terminate with it. *U_threads* may issue Unix system calls.

#### *4.2.5 Reconfiguration*

The reconfiguration capabilities of CHORUS/MiX may be summarised as follows :

- In the case of node failure, it must be possible to reconfigure the interrupted application on a backup node in order to recover its full functionalities.
- In the case of upgrading a running application with a new version of one of its component, the replacement of that component may leave the overall application unchanged. The new version may lead to the migration from one node to another of that new component; consistency of the whole application must be saved.
- In the case of dynamic load balancing, or network extension, the replacement or the migration of some element toward a new location must not introduce any disruption in the executing software.

CHORUS provides a basis for supporting efficient dynamic application reconfiguration. The key point is the flexibility of the inter-process communication provided by distribution transparency, ports and groups of ports.

## 5 Porting CHORUS/MiX onto the H1

Porting the CHORUS/MiX operating system onto a new hardware architecture is a two-phase process.

First, the Nucleus must be adapted to the new hardware 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. For keeping the Nucleus highly portable, the hardware dependencies have been isolated in small and well-defined components: the *Supervisor* and the *machine dependent memory management module.*

Second, the hardware dependencies of the System Servers implementing the Unix Subsystem must be modified in the same way. As for the Nucleus, System Servers have been designed in such a way to allow porting with minimum effort.

### *5.1 Porting the Nucleus*

The basic abstractions implemented and managed by the CHORUS Nucleus are shown in Table 1. The porting effort of these abstraction is principally focused on the thread and region objects.

The file system is extended with the Network File System (NFS®). NFS is a facility for sharing transparently files in a heterogeneous environment of machines, operating systems, and networks. CHORUS/MiX processes from any node in the H1 network, may access files localised on another machine's disk through ETHERNET.

CHORUS/MiX supports the standard X Window System® facility. An X client on a site of the H1 network may execute a graphic system call to an X server connected to a LAN. This feature allows a user to use an X-terminal to work on an H1 machine under CHORUS/MiX .

The CHORUS/MiX operating system defines a true Unix interface and hence allows a extensive code portability.

## *4.2 Managing the Distribution*

CHORUS provides extensions to the Unix interface in order to take benefit from the distributed nature of the system and the underlying CHORUS Nucleus services such as IPC or Threads. Such accesses are not provided by invoking directly the Nucleus, but through Unix system calls, in order to eliminate inconsistencies.

### *4.2.1 Process Management*

The basic extension of process management deals with enabling remote creation (*fork*) or execution (*exec*) of processes. A "creation site" information has been added to the system context of Unix processes. It may be set through a new system call : *csite(SiteId)*. This information is used when a process is *fork*'ing or *exec*'ing : on a fork, the child process will be created on the site specified by *SiteId*. On an *exec*, the process will start the execution of the new program on the site specified by *SiteId*.

### *4.2.2 File System Extension*

The naming facilities provided by the Unix file system have been extended, to permit the designation of services accessed via Ports.

Symbolic Port Names, defined as a new Unix file type, can be created in the Unix hierarchical file system. They associate a file name to a port identifier. When such a name is found during the analysis of a pathname, the corresponding request is forwarded to the port.

User written servers as well as system servers can be designated by such symbolic port names, thus allowing "users" to dynamically bring some extensions to the system. This facility is used to interconnect file systems of different sites and provide a global name interface.

### *4.2.3 Inter Process Communication*

Unix processes can create ports, insert ports into groups, and send and receive messages. They can migrate ports from one process to another. CHORUS IPC mechanisms allow them to communicate transparently over the network. Applications can therefore be tested on a single machine, and then distributed throughout the network, without any modification necessary to adapt it to a new configuration. Using port migration or

---

® NFS is a trademark of Sun Microsystems, Inc.

® The X Window System is a trademark of the Massachusetts Institute of Technology.

The user can execute its application on the network of H1s transparently from the multiprocessor nature of the machine. Distribution may span several networks of H1s managed by CHORUS/MiX with the same level of transparency. In other words, the configuration of the application is managed as flexibly as hardware configurations.

## *4.1 A Standard Unix*

CHORUS/MiX controlling a network of H1s defines a multi-processor machine managed through a standard Unix environment. It supports a transparent distributed file system, tools for software development applications, support for standard interprocess communication and local area networking.

### *4.1.1 Unix Interfaces*

CHORUS/MiX provides the user with a Unix environment composed of:

- one or several file systems accessible transparently from each site;
- a command interpreter of his choice;
- a large set of Unix commands: the whole set of the Unix System V commands is available;

The user logged on a site of the H1 network finds all the tools he is used to find in his environment, he is able to run each command from any node in the network. The programming interface is compatible with the Unix System V. To develop a program, the user will use standard and well-known primitives and features:

- *open*, *read* or *write* operations to access files;
- *fork*, *join* operations to manage Unix processes;
- pipes to communicates between these processes;
- signal management (*kill*, *signal*, ...).

The user may execute a call to these primitives from any node wherever in the network that node is located. The compatibility and transparency of objet and service location allow CHORUS/MiX to support a large set of tools developed for the Unix operating system (*sed*, *awk*, *lex*, *yacc*, ...).

### *4.1.2 Unix Enhancements*

Some features have been added to extend the Unix System V. These enhancements, mainly issued from the BSD systems, offer services for network management and communication.

CHORUS/MiX fully supports the socket-based inter-process-communication facilities adapted from BSD. Messages and packets may be exchanged under the socket protocol between CHORUS/MiX sites or between a CHORUS site and a heterogeneous site on a local area network.

---

® ETHERNET is a trademark of Xerox Corporation.

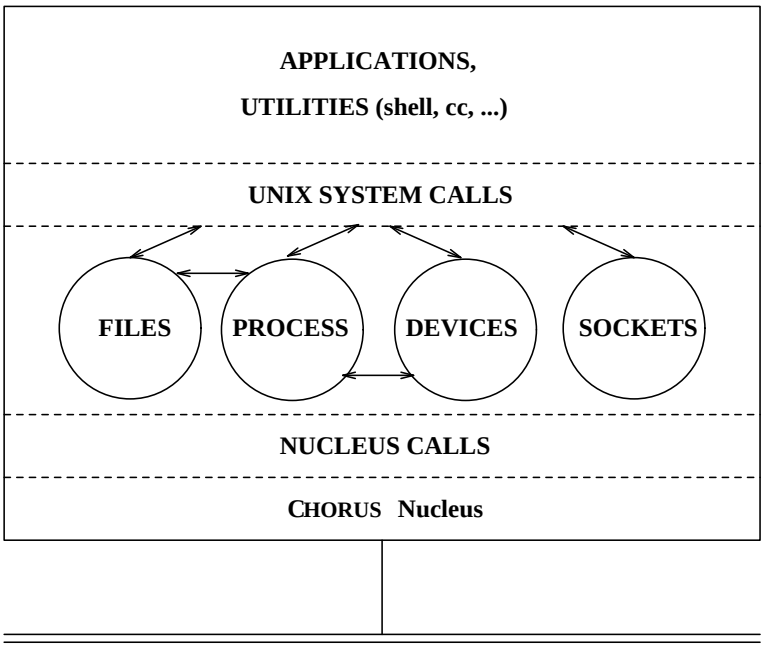


Figure 4: The CHORUS/MiX subsystem

several significant extensions which offer, at the Unix subsystem level, access to CHORUS facilities.

Powerful real-time facilities available at the Unix subsystem level make it possible to dynamically connect handlers to hardware interrupts. Unix processes enjoy the benefit of priority-based preemptive scheduling provided by the CHORUS Nucleus, possibly at a higher priority than the CHORUS/MiX subsystem servers themselves.

Unix processes running on CHORUS can communicate with other Unix processes, bare CHORUS processes or entities from other subsystems using the CHORUS IPC mechanisms. Processes, for example, are able to create and manipulate CHORUS ports, send and receive messages, and issue remote procedure calls.

### *3.6 User Defined Servers*

The CHORUS architecture provides a powerful, convenient platform for operating system development. The homogeneity of server interfaces provided by the CHORUS IPC allows system builders to develop new servers and integrate them at will into a system, either as user or as system servers. For example, new file management strategies such as real-time file systems, or fault-tolerant servers can be developed and tested as a user level utility without disturbing a running system, using the powerful debugging tools available for user-level application development. Later, the server can be migrated easily within the system for performance consideration.

## 4 CHORUS/MiX on the H1

The CHORUS/MiX system is based upon the Unix operating system as derived from the Unix System V. It also includes features of BSD 4.3 system, and other CHORUS/MiX enhancements. A Transputer machine under CHORUS/MiX from a user point of view is defined by a network of H1 processors, one or more disks, an ETHERNET® interface, and some link interface for specific I/O.

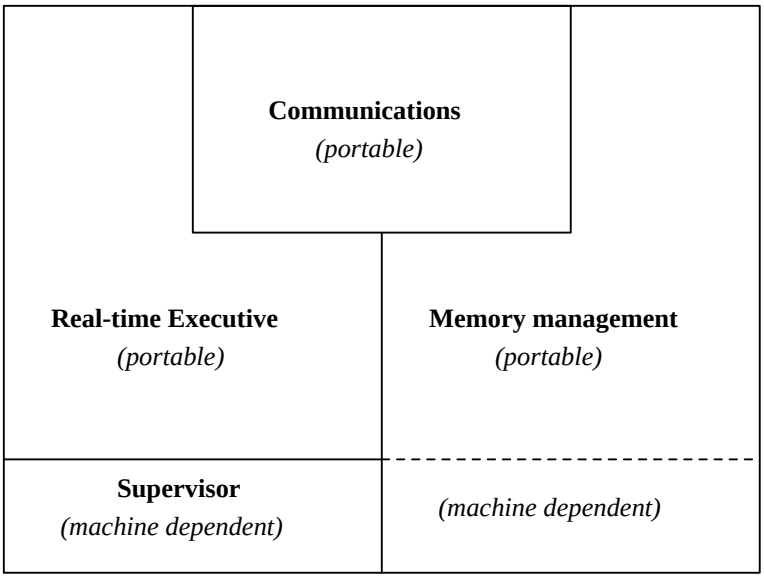


Figure 3: The CHORUS Nucleus

the Nucleus. Their position "on top" of the Nucleus provides a structured, well-defined way for system builders to cope with complexities of operating system development.

In the operating system context, a subsystem offers the means by which to supply a complete standard operating system interface to standard application programs. New servers can then be added to the set of servers delivering this interface as a means of gracefully extending system and operating system capability. *Servers* are the building blocks, the toolset, that system builders use to incorporate new distributed functions.

## *3.4 The CHORUS/MiX Subsystem*

CHORUS/MiX (see Figure 4) integrates a Unix System V subsystem with the CHORUS Nucleus to provide a standards-based, real-time, transparently distributed Unix environment. CHORUS/MiX Unix services conform to X/Open specifications and have been extended to real-time and to the distributed environment (distribution of programs as well as files), all in a way that is completely transparent to Unix application programs. A CHORUS/MiX that conforms Unix System SVR4, with real-time and distributed features, is under development.

CHORUS/MiX offers the traditional set of Unix functions for managing signals, but extends them to manage real-time, multi-threaded and distributed processes. This extension permits the creation - and manipulation from a distance - of processes on any machine, while respecting rules of Unix regarding environments, open files and so on.

Finally, because CHORUS/MiX is fully compatible with and actually *contains* Unix System V source code, it allows Unix processes to take advantage of such CHORUS facilities as multi-threading, location transparent IPC, virtual memory management, device drivers and servers development in user space and more.

## *3.5 Real-time and Distribution in CHORUS/MiX*

The modular design of CHORUS/MiX allows binary compatibility with Unix , while maintaining a well-structured, portable and efficient implementation. It also includes

The immediate domains of application for CHORUS products are those for which the limits of traditional operating systems are preventing the development of fully satisfactory solutions. This includes, for instance, the development of standard operating system on advanced hardware architectures, in particular parallel machines like Networks of Transputers.

## *3.1 Basic Structure - Nucleus and Subsystems*

The CHORUS architecture is based on a minimal real-time distributed *Nucleus* that integrates distributed processing and communication at the lowest level. The Nucleus provides generic tools - thread scheduling, real-time event handling, network-transparent inter-process communications (IPC) and memory management - for independent servers called *subsystems*, which coexist on top of the Nucleus.

Subsystems separate the functions of the operating system into sets of services provided by autonomous servers, and provide operating system interfaces to application programs.

The CHORUS organisation of Nucleus and subsystems represents the most logical view of an open operating system for cooperative computing environments. Separating functions increases the modularity, portability, scalability and "distributability" of the overall system, which has a small, trustworthy micro-kernel as its foundation.

## *3.2 The Nucleus*

The CHORUS Nucleus (see Figure 3) provides both local and global management of services. At the lowest level, it manages the physical resources at each "node" with four clearly defined components:

- a classical *real-time multi-tasking executive* which controls allocation of local processors, manages priority-based preemptive scheduling of CHORUS *threads*, and provides primitives for fine grain synchronisation of, and low-level communication between, threads;

- a *distributed memory manager* which can support the full range of memory architectures - linear, segmented or virtual;

- a low level hardware *supervisor* which dynamically dispatches external events such as interrupts, traps and exceptions to dynamically defined routines or ports;

- an *Inter Process Communication* (IPC) manager provides the high-performance global communication services (exchange of *messages* through *ports*) which are the key to CHORUS architecture's real-time, distributed capabilities.

## *3.3 The CHORUS Subsystems Concept*

Subsystems in CHORUS architecture are sets of system servers that use the generic services of the Nucleus to provide higher-level services. More simply, a subsystem is an operating system built on top of the CHORUS Nucleus.

Subsystems manage physical and logical resources such as files, devices and high-level communication services and they communicate via the *IPC* facility provided by

### 2.3.2 A Rationalised Scheduler

The interface to the scheduler has been cleaned-up and rationalised to allow easy manipulation by real-time kernels or operating systems. The first rationalisation involves the manipulation of an interrupted process, of particular importance to real-time kernels implementing fast interruptible handlers.

When the scheduler swaps from a low-priority to a high-priority process, the state of the interrupted process is saved in a shadow copy of the processor's registers. This copy can be saved to/restored from memory using special instructions. With this mechanism, the H1 allows clean manipulation of its internal state, enabling improved software scheduling on interrupts.

Another aspect of this improved interface is the availability of atomic instructions to modify the hardware scheduler queues, enabling operating systems and real-time kernels to use the lightningly fast context switching time of the Transputer, while maintaining many more levels of process priority than the two offered by the H1 hardware.

Finally, semaphores, first introduced by Dijkstra in 1965 as a useful mechanism for controlling access to shared resources by concurrent processes, have been included in the H1 features. Two new instructions, `signal` and `wait`, have been introduced to atomically handle semaphore structures allocated in memory. This feature greatly simplifies the design of software kernels, as many processes usually need to gain access to shared kernel information.

# 3 CHORUS

The CHORUS family of operating systems is centered around the small real-time distributed CHORUS *Nucleus* which provides a minimal set of essential generic services. The other members of the family are built on top of the Nucleus and inherit its real-time distributed computing capabilities. The CHORUS/MiX operating system incorporates the CHORUS Nucleus as a base on which it builds a Unix interface that is transparently extended to distributed processing and to use in real-time environments.

The key characteristics of the CHORUS family of operating systems are:

- *Real-time*: CHORUS systems are built on the real-time CHORUS Nucleus and have all the functional characteristics and performance of classic real-time executives.

- *Controlled transparent distribution* of processing and of data: the management of distribution can be entirely taken care of by CHORUS systems, yet controlled by network administrators. In particular, CHORUS permits dynamically reconfiguring the system and applications.

- *Modularity*: CHORUS systems are composed of a set of communicating modules that can be assembled and reconfigured dynamically depending on the hardware capability and application needs.

- *Openness*: CHORUS/MiX provides all the functionalities of a standard Unix system (Unix SVR3.2 today, SVR4 in the future).

- *Compatibility* with Unix is provided by CHORUS/MiX : application programs, utilities, and files used under Unix are supported without modification (binary compatibility) by CHORUS/MiX .

- At the destination side, a c104 delivers the packet to the appropriate H1 link, where the VCP will reconstruct the message from all the successive packets associated with the appropriate channel.

Providing two separate chips for computing and routing offers many advantages:

- For small networks with only neighbour communications, H1s can be used without IMS C104, sparing the silicon cost of on-chip routing for other H1 functionalities.
- Routers can have many links (32 for the IMS C104), minimising the number of routers in a network, and hence minimising the routing delay.
- Since messages do not flow through the H1s the total link bandwidth required is just the one required by local processes.
- Network structure and scalability is independent of the number of processing nodes.

### *2.2.3 H1 Communication Capability and CHORUS*

The high flexibility and performance of the H1 communication system enable CHORUS ' Inter-Process-Communication system to be implemented with an unprecedented efficiency, to an extent where local memory bandwidth becomes more of an issue than communication bandwidth.

## *2.3 Enhanced Process and Scheduler Models*

Improved performance and communications are not the only aces dealt to the H1 in its cradle. The process model itself has been refined and the interface to the scheduler rationalised.

### *2.3.1 Process Behaviour Control: the Dual Processes*

A new type of process, and a new mode of execution, have been added to the features of H1. The new process type, or *L-process*, offers local handling of error conditions, debugging traps, and a special instruction for implementing operating systems "system calls". The new mode of execution, or *P-mode*, allows an L-process to run in a protected virtual address space.

Both P-mode and L-processes behave in a very similar way. In each case, the execution stream switches to another process (the L-process *stub* for the P-mode, a *trap-handler* for the L-process) whenever an exceptional situation is reached. This other process, or controlling process, deals with the exceptional situation on behalf of its dual. Thereafter, the initial process can be continued or discarded depending on the seriousness of the exceptional situation.

The two main differences between P-mode execution and trap-handling lie in the reduced instruction set available to P-mode (i.e. no scheduling or communication instructions) and the execution of P-mode processes in a segmented and protected virtual address space.

to run its links at 100Mbits/s, giving a global bidirectional bandwidth of 80 Mbytes per second.

### *2.2.2 The IMS C104: a Generic Routing Chip*

The H1's VCP enables any number of channels to link the Transputer with the external world, but this is not enough to allow direct communication between any two processors in a network. This is the job of a specialised chip, the IMS C104.

The IMS C104 is a general purpose worm-hole packet-switched routing chip, whose block diagram is shown on Figure 2.

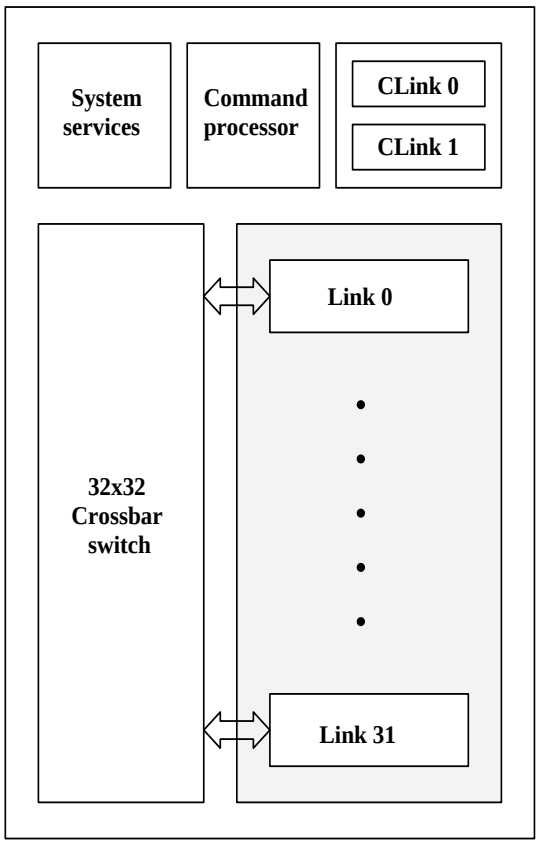


Figure 2: IMS C104 block-diagram

It cooperates with the H1 in the following way:

- Any message sent by an H1 over a virtual channel is split into a number of small packets by the VCP. This will enable the VCP to fairly multiplex physical links by multiplexing the packets. Each packet contains a minimal header describing the destination of the message.

- The links of the H1 being connected to a IMS C104, the packets are routed to their destination encoded in the header. Worm-hole routing means that the routing of a packet takes place as soon as the header has been received by the IMS C104, potentially before the whole message has entered the chip. This mechanism makes routing extremely fast; less than a micro-second is spent between reception of the first bit of a packet and its retransmission out of the IMS C104.

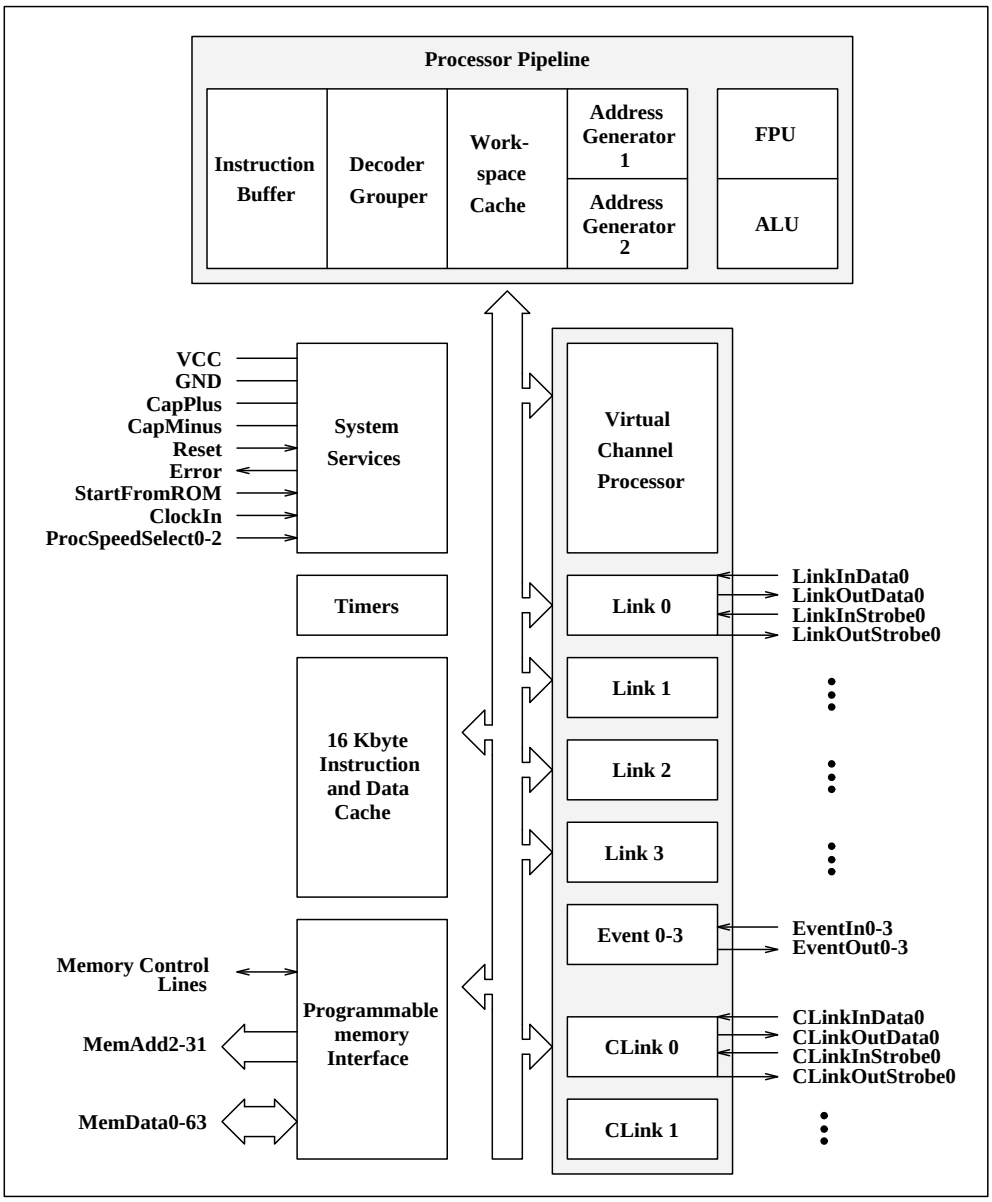


Figure 1: H1 block-diagram

channels can be handled by a network and each processor can have access to up to 65536 of them.

### *2.2.1 The H1's Virtual Channel Processor*

The H1's VCP is the piece of hardware which handles all off-chip communications. Its main role is to multiplex all the outgoing and incoming communications over the chip's four physical links. The VCP makes this multiplexing totally transparent to the CPU. As far as the programmer is concerned, there is a quasi-infinite number of communication channels going off-chip. Which physical link they will use is only known when the VCP is configured, before any application starts executing.

The fact that the VCP is a separate entity from the CPU also means that computation and communication can proceed entirely in parallel, overlapping each other.

To sustain the high communication bandwidth required by the VCP, a new electrical protocol has been devised for the links; the "Data/Strobe" mechanism enables the H1

## 2 The H1: A Virtual Machine

INMOS® Transputers are complete microcomputers integrated in a single VLSI chip. Each Transputer combines an integer ALU, a two-priority hardware scheduler, some on-chip memory, an advanced external memory interface and communication links. In addition, some versions incorporate a floating point unit or some application specific logic. The strength of the Transputer lies in the close integration, both at the micro-architecture and software level, of communication and (multi-)processing. This makes the design of parallel or concurrent systems using Transputers especially easy and efficient.

The H1 is the first member of INMOS' new generation of Transputers. After achieving "virtual processing" on the first generation (the hardware scheduler gives each process access to a "virtual sequential processor"), INMOS enters the age of "virtual communications" with the new generation. But the improvements over the first generation Transputer go much beyond enhanced communication.

### *2.1 Performance Improvements*

The IMS T8xx® family of Transputers offered $\approx$ 10 Mips and $\approx$ 3 Mflops peak performance. The H1 boosts these values to more than 150 Mips and more than 20 Mflops ($>$ 60 Mips and $>$ 10 Mflops sustained) while staying 100% compatible with the IMS T805 instruction set.

To achieve this amazing result, INMOS went through a careful redesign of the micro-architecture. The H1 has a totally new pipelined superscalar architecture, allowing it to execute many instructions grouped together every clock cycle. This pipeline is itself driven by a high speed clock: while retaining the Transputer 5Mhz external clock, the first H1 will be internally a 50 Mhz device.

To feed its powerful pipelined execution units, the H1 includes a two-level cache offering access to multiple memory locations each cycle. The first level cache, organised as a multi-ported register set, gives concurrent access to the memory locations in the current process' work-space. The second level is a generic 16Kb fully associative cache. This main cache replaces the on-chip memory found on earlier Transputers.

All these features should make even single H1 nodes one of the fastest CHORUS/MiX machines on the market. Figure 1 shows the cache and pipeline described earlier.

### *2.2 Communicating through Virtual Channels*

Communication and parallelism have always been key features of INMOS processors. Mechanisms are offered, at the instruction set level, for building parallel programs of communicating processes. The first generation of Transputers offered intra-Transputer communication through memory to memory DMA and inter-processor communication through DMA over serial communication links. The **input** and **output** instructions gave direct access to the underlying hardware running at 20Mbits per second.

Though enabling very efficient parallel machines and programs to be built, this initial design induced undesirable limitations on some application programs.

To lift all these limitations, INMOS introduces virtual channels on the H1 family of Transputers. A virtual channel enables communication between any processes in a network of processors, irrespective of their physical location. Any number of virtual

# CHORUS on H1: UNIX System V on Networks of Transputers

Dominique Grabas[***], Marc Guillemont[**], Michel Tombroff[**], Christian Tricot[*] and Hossein Yassaie[***].

[*] *ARCHIPEL, 4 route de Nanfray, Cran-Gevrier, 74000 Annecy, France.*
[**] *Chorus systèmes, 6 av. G. Eiffel, 78182 Saint-Quentin-En-Yvelines, France.*
[***] *INMOS Limited, 1000 Aztec West, Almondsburry, Bristol BS12 4SQ, UK.*

**Abstract**

This paper describes how the CHORUS distributed operating system is adapted to the H1 Transputer. It shows how CHORUS will take advantage of the unique H1 features for communication and distribution and how, in turn, CHORUS/MiX will provide on the H1 a standard Unix environment where distribution and parallelism are carefully integrated in order to extend Unix capabilities. The user's view of the system as well as the main lines of its implementation are highlighted.

## 1 Introduction

The Transputer family of processors has integrated the communication as a basic function of the processor. CHORUS ®, on the other hand, is the first commercially available product which introduces communication as a basic function of a standard operating system. How could they miss one another?

Having the standard UNIX®offered by CHORUS available on the H1 Transputer opens the field for new applications and new markets, reconciling the benefit of standard interface and the potential of efficient parallelism.

ARCHIPEL, Chorus systèmes and INMOS, a member of SGS-THOMSON Microelectronics Group, conduct a joint development program to port the CHORUS operating system to the H1 Transputer.

This paper describes CHORUS on the H1 Transputer, how it will be perceived by its users and how it is built.

---

® CHORUS is a registered trademark of Chorus systèmes.
® Unix is a registered trademark of UNIX System Laboratories, Inc. in the U.S.A. and other countries.
® ARCHIPEL is a registered trademark of ARCHIPEL S.A.
® INMOS and IMS are registered trademark of INMOS Limited.