G. Courcoux, P. Vittone

Porting OmniPCX 4400 to Windows NT and Linux

> When changes in a telecommunication system lead to a change of operating system, rigor and method are essential.

Introduction

The OmniPCX 4400 is an Alcatel PBX designed for use in medium to large installations, or for companies that require a wide range of services. It can handle 50 to 5 000 users, or up to 32 000 on a private network; it offers virtual PBX services in a private network, call center services, communications compression services, voice over IP services, PC telephony services, etc (see *Figure 1*). The services it offers meet current market demands, and even anticipate future needs.

Nevertheless, in the dynamic world of telecommunication, the PBX market faces constant evolution: the integration of voice and data networks, the integration of diverse communication services, information management and productivity applications, and hardware platform changes, to mention only the most important. For some of these changes, the choice of operating system may be important. It must, for example, rapidly take into account changes in hardware platforms, IP protocols, etc. Similarly, standard interworking mechanisms between applications from various origins (CORBA, COM/DCOM, etc) must be available, as must easy to use development tools. As far as possible, we should not become involved with the operating system, so that we can concentrate on providing added value for our customers.

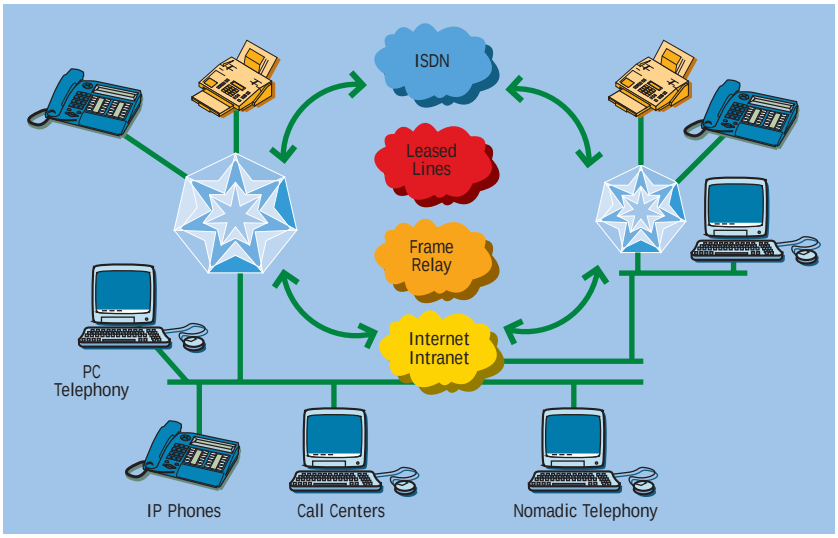

Figure 1 – Some OmniPCX 4400 services

Finally, the operating system influences the opinion of our customers concerning the long life, reliability, ability to evolve and openness of our products.

All these considerations have naturally led us to want to use operating systems that are well supported and widely recognized by the market. Windows NT meets these criteria. It is the *de facto* standard for the information system market. Most applications are available for Windows NT, which is or will be used by an increasing number of telecommunication systems. It is perfectly suited to the installation of the communication applications server which brings together all the Alcatel and third-party applications ... a server for which the PBX functions are only a part of the whole.

We decided to use Windows NT for certain future versions of the OmniPCX 4400. This made it necessary to verify that its characteristics could meet our needs, and to define a suitable Migration path.

As for Linux, the speed with which it has been adopted by many enterprises, its development dynamics, and the quality of its support ensure that it will soon meet our criteria. It is already reliable, free and requires few memory resources, as well as offering an extensive range of services. We decided to test our software with Linux, to validate not only the migration methods we have defined for porting to Windows NT but also Linux's characteristics, which led us to use it in other Alcatel projects.

Current OmniPCX 4400 Software

Operating System

The OmniPCX 4400 at present runs under Chorus/MiX, an operating system designed in the late 1980s by Chorus Systems (today a part of Sun Microsystems). This system, which was ahead of its time with its micro-kernel architecture and distribution capabilities, manages two worlds with very different behaviors: a real-time world in which applications are programmed using the Chorus Application Programming Interface (API) and a Unix world that implements AT&T's Unix System V API.

The Chorus API handles kernel objects such as processes, tasks, task synchronization objects, messages, ports and groups of ports for Inter-Process Communication (IPC), as well as virtual memory management. This API is used by applications with real-time operating constraints, such as telephone extension management.

At Alcatel's request, Chorus Systems extended the Unix System V standard API by exporting part of the Chorus API. As a result, Unix processes can become multi-threaded, control their scheduling characteristics, modify their virtual memory attributes and take advantage of Chorus IPC.

Finally, integrating Chorus/MiX into the OmniPCX 4400 led us to change the IP protocol stack and enrich the system with various device drivers, system calls, etc.

Software

The OmniPCX 4400 software, which is written in C/C++, comprises well over 5 million lines of code. The seven real-time applications, which represent 35% of the code, process subscriter calls using only the Chorus API and a small number of functions from the standard C library.

The remainder of the code corresponds to over 200 Unix applications providing numerous functions, such as billing, traffic monitoring, configuration, directory, application and equipment status analysis, maintenance, etc. They use the Chorus, Unix System V and BSD socket (for IP network access)APIs.

Finally, we use the Bourne *shell* as our Unix system command interpreter; we have written several hundred scripts in the language specific to this interpreter.

Production Management

Such a large quantity of software requires sophisticated follow-up and production tools to manage various versions in parallel – both maintenance versions and major product changes.

Code is currently produced on Sun Sparc machines running Solaris with GNU C++ compiler, the widely distributed public domain compiler.

We use a large number of tools to track software changes and correct problems, carry out specific processinges before compilation, build in parallel on multiprocessor machines, distribute the production load, monitor production, etc.

Some of these tools are available only for Sun Solaris. Others could be ported to this system, but this would require considerable effort.

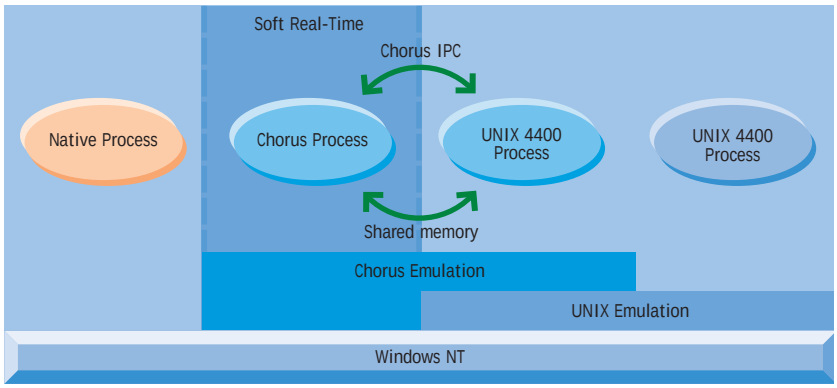

Figure 2 – Various types of processes on Windows NT

Migration Methods

Migration methods have been predefined because of the strength of the existing constraints. These methods must make it possible to conserve the source code, which is largely unique for all operating systems, since Chorus/MiX and Windows NT have to coexist. It would be inadmissible to have to repeat the development several times. Moreover, several million lines of code cannot be extensively modified at a reasonable cost and without risking serious regressions.

Consequently, it is clear that the only solution is to supply all the operating systems, using emulation where necessary, with the various APIs that are currently in use. We chose this approach for the Chorus, System V, and BSD *socket* APIs.

In the case of Windows NT, our Unix applications must be based on a Unix emulation whici provides the main Unix commands as well. *Figure 2* shows the various processes that must exist on Windows NT and their interactions with the emulations.

Finally, the compiler must run on Sun Sparc machines under Solaris to be able to use the current tools. It is therefore a cross-platform compiler.

Porting to Windows NT

Windows NT's reputation as a real-time operating system has not been completely established. Therefore, we tried to assess its potential and take into account its limitations.

Real-Time under Windows NT

Several studies regarding the possibility of using Windows NT as a

real-time operating system are available on-line [1,2]. Microsoft also supplies documents containing various recommendations. In all cases, the authors acknowledge that the Windows NT architecture is not adapted to so-called *hard* real-time, that is, it cannot handle interrupts within a predetermined time.

There are several real-time extensions for Windows NT that meet this need. These extensions modify the Windows NT Hardware Abstraction Layer (HAL) to receive and process the interrupts. Processing that takes place in real-time always has a higher priority than Windows NT processing, including processes associated with NT driver interrupts. In addition, at best the real-time part only has extremely restricted access to Windows NT services and must use specific mechanisms to communicate with NT processes. Moreover, HAL modifications mean that not all hardware is managed and these extensions must evolve at the same pace as the hardware. Finally, these extensions are often strongly linked to Microsoft's Visual C++ compiler and their cost is quite high, both of which are disadvantages.

On the other hand, Windows NT is fairly well adapted to support *soft* real-time, where processing must almost always take place within a predetermined and relatively long timeframe. For this type of processing, the operating system must at least have the following features:

- ability to set task priorities;
- pre-emptive scheduling based on priority;
- low latency for high priority applications, even when the hardware is carrying a high load of lower priority applications;
- task synchronization and inter-process communication mechanisms;
- possibility of locking real-time process pages in memory;
- absence of priority inversion when accessing the above services and hardware.

In addition, the existing software requires mechanisms for sharing memory between processes.

The study showed that Windows NT meets the needs of *soft* real-time processing fairly well, except for the absence of priority inversion. We have developed a mechanism (see *Figure 3)* for detecting these inversions, which we noticed are caused by most Win32 API functions. Nevertheless, secure functions, from this point of view, were sufficient to develop the Chorus API emulation. This is the approach we chose.

To avoid having to use Windows NT's real-time extensions, we have modified our architecture in order to only have to process signaling messages transmitted over IP. This architectural change enhances flexibility by separating the operating platform from interface equipment, allowing standard platforms to be used. As for signaling messages, they must be processed within a time that is acceptable to the user but sufficiently long for the processor, that is, within 50 or 100 ms. We are therefore well within the framework of *soft* real-time.

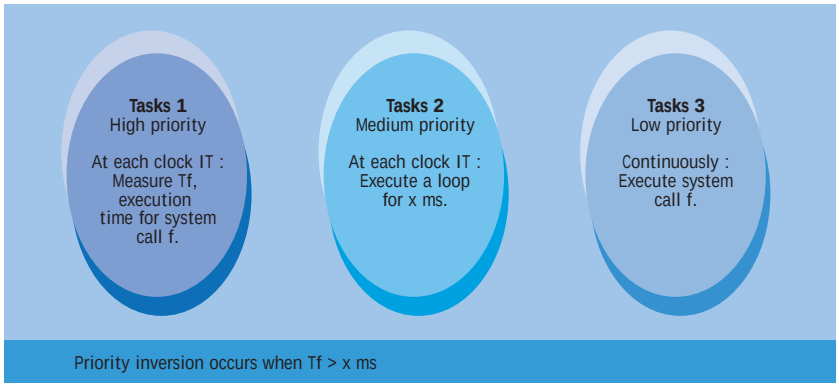

Figure 3 – Priority inversion detection

Chorus Emulation

Our Chorus emulation takes into account the constraints of Windows NT, in particular the presence of priority inversion. So, it only uses the primitives for memory management, task creation and output, task and process priority positioning and synchronization by signaling. There are no named objects associated with Chorus objects. We use a proprietary IPC mechanism via shared memory. Two high priority servers supply all services; one of them is dedicated to memory management.

However, priority inversions remain in memory management and task creation operations. They are directly linked to the way the Windows NT kernel and the drivers are written. Nevertheless, since these operations only take place at start-up, they are not a problem.

Unix Emulation on Windows NT

Developing a Unix emulation involves a significant amount of work, so we decided to use and distribute an existing emulation.

There are a number of Unix emulations running under Windows NT; all offer a large number of system calls and Unix libraries as well as the main Unix commands. In our case, the Unix emulation must interact with the Chorus emulation, which implies that the Unix emulation source code has to be modifiable and that an application based on this emulation should be able to use the Win32 API. It must also be reasonably priced.

These criteria led us to choose CYGWIN, by Cygnus Corp, because of the availability of the source code and its cost. Nevertheless, as CYGWIN does not support multitasking processes, we will develop this function.

Compiler

At first, Cygnus Corp offered support services for public domain

GNU software, mainly the compiler. They developed CYGWIN to port the GNU compiler to Windows NT, which was simultaneously modified to produce executables for this operating system.
Since we are now using this compiler for the current product, we have decided to test the modified compiler for Windows NT. On this occasion the overall product's good quality was noted, in particular to take into account different environments, since we had no problem in obtaining and using a Sun-based cross-platform compiler producing for Windows NT, a configuration not officially supported by Cygnus Corp.

Moving to Windows 2000

At all levels, both in the Chorus emulation and in CYGWIN, as well as in our development of Windows NT specific applications, we use the Win32 API, Microsoft's C library and, for some applications, the Microsoft Foundation Classes (MFC). All these elements have been retained with a high degree of compatibility in Windows 2000. Thus the move to Windows 2000 will be easy. CYGWIN, for example, already operates in this environment.

Porting to Linux

Specific Constraints

To realistically test our software on Linux, we must take into account the additional constraints compared with porting to Windows NT. In contrast to OmniPCX 4400/NT, which uses a standard hardware platform, OmniPCX 4400/Linux uses a CPU card plugged in the OmniPCX 4400 backpanel, which is strictly identical to the one currently used. For cost optimization reasons, processor resources and CPU memory are limited compared to those on today's personal computers.

Consequently, we must take the greatest care in creating the various emulation layers necessary so that we don't waste the improved performance, from the Unix point of view, that Linux offers.

Linux and Real-time

If the Chorus micro-kernel has better real-time characteristics than the Linux kernel, the latter offers three scheduling policies and exports a subset of the Portable Operating System Interface (POSIX) API, bringing it up to *soft real-time standards*.

The Linux kernel scheduler uses priorities and guarantee that the next task to be executed will always be the one with the highest priority. On the other hand, it can only pre-empt tasks running in user mode. An advantage arising from this weakness is that it does not introduce the priority inversion phenomenon. However, certain lengthy operations in the kernel may lead to the current task keeping the processor for more than 200 ms, while a higher priority task is waiting to be processed. This phenomenon, called scheduling latency, is caused by a critical section being too long.

Chorus/MiX had a very low latency since this critical section was restricted to calls from the micro-kernel, which have a very limited execution time compared with that of a monolithic Unix type kernel. Nevertheless, it can be noted that this drawback will disappear in the medium-term, as Linux is more widely used in multiprocessor architectures. This type of architecture led Linux's programmer to re-examine the data protection policy in the kernel, based on locks, by forcing the creation of more locks with a shorter use time. Nevertheless, such modifications may lead to priority inversion phenomena.

Initially, a real-time extension could serve as a stopgap measure. Currently, the two main ones are Real-Time Linux (RTL) [3] and Real-Time Application Interface (RTAI) [4]. RTL was developed by the University of New Mexico, and RTAI by the Aerospace Engineering Department at the Milan Polytechnic Institute. These extensions are free and actively follow the developments of the Linux kernel, thus guaranteeing a certain longevity. The problem, already noted with Windows NT real-time extensions, lies in the reduced API and the constraints involving means of communications with Linux processes. This is aggravated by the fact that these two extensions only offer real-time characteristics to privileged code, that is code that can modify everything in memory without hindrance. These extensions are clearly inappropriate given the quantity of code (35% of the total volume) subject to real-time constraints.

A promising development has recently emerged from the professional music world. It significantly shortens the scheduling latency to about 1 ms, which is compatible with the OmniPCX 4400 requirements.

Chorus Emulation

Chorus emulations under Windows NT and Linux will have fundamentally different implementations, since each of them directly handles operations specific to each system, maximizing their respective advantages.

Moreover, this emulation is used by the real-time part of the OmniPCX 4400: it must have a very high performance. To this end, it will be developed as a module independent of the kernel, which is loaded during the system start-up phase. The cost of communication between real-time applications and emulation will thus be minimized, and the data associated with Chorus objects will always be present in memory, as is the case with OmniPCX 4400 today.

Unix Emulation

Talking about Unix emulation on a Linux platform may seem somewhat strange. In our particular environment, the goal is to minimize conditional compilation orders in the application source code, therefore exporting an API resembling the one used by the 200 Unix applications in the OmniPCX 4400. Fortunately, this API is quite similar to the native Linux API, which will make it possible to optimize its development.

The extensions necessary for this API, like the one related to multitasking aspects, were created under Windows NT using a subset of the POSIX API, which is itself emulated

under Windows NT. Using this method greatly facilitates porting from Windows NT to Linuxof the emulation of these functions, typically of Chorus/MiX from Windows NT to Linux.

Production Line

The Linux system already uses the GNU production toolkit to produce everything from the kernel to tools and libraries. The general practice in the Linux world is to generate all software natively, that is to say, on a Linux machine. Cross-platform production experience has already proved that migrating the generation of the kernel and several tools to a Solaris machine is straightforward.

Source Code Availability

The availability of the Linux source code is an advantage for controlling the entire system and allows us to repair observed malfunctions in a very short timeframe.

When Linux is used the framework of a non-distributed test, the problem of licenses (mainly the GNU Public License) for the source code does not apply. Of course, in its commercial use of this operating system, Alcatel will strictly comply with the licenses and will try to contribute to the improvement of Linux by distributing details of any problems encountered and the proposed solutions.

Conclusion

Changes in telecommunications modes and methods require major changes in products. In the case of OmniPCX 4400, which offers a high level of service, these changes may not involve abandoning existing elements.

It is therefore absolutely necessary to establish methods that facilitate the evolution of telecommunications products, in particular their software. In the case of a change of operating system, these methods lead us to use powerful emulation suited to the new operating system.

The methods discussed here enable Alcatel to optimize the use of its development efforts. They also guarantee the longevity of our customers' investments.■

References

1 "Using the Windows NT Operating System for Soft Real Time Control – Separating Fact from Fiction, http://www.icsmagazine.com/Lit/9808-A_B.html.

2 "An Empirical Evaluation of OS Support for Real-time CORBA Object Request Brokers", http://www.cs.wustl.edu/~schmidt/RT-OS.ps.gz.

3 Linux RT Home Page http://www.cs.nmt.edu/~rtlinux/.

4 RTAI Home Pagehttp://www.aero.polimi.it/projects/rtai.

Gilles Courcoux is the operating system architect for the OmniPCX 4400; he is in charge of the OmniPCX 4400 test under Linux at the Alcatel Enterprise & Consumer Group in Colombes, France.

Pascal Vittone is in charge of porting the OmniPCX 4400 onto Windows NT; he is located at the Alcatel Enterprise & Consumer Group in Colombes, France.