1-1992

Porting Chorus to the PA-RISC: Project Overview

Jonathan Walpole
Oregon Graduate Institute of Science & Technology

Marion Hakanson
Oregon Graduate Institute of Science & Technology

Jon Inouye
Oregon Graduate Institute of Science & Technology

Ravi Konuru
Oregon Graduate Institute of Science & Technology

Citation Details

Walpole, Jonathan, Marion Hakanson, Jon Inouye, and Ravindranath Konuru. Porting Chorus to the PA-RISC: Project Overview. Technical Report CSE-92-3, Oregon Graduate Institute, 1992.

Porting Chorus to the PA-RISC: Project Overview

Jonathan Walpole,
Marion Hakanson,
Jon Inouye,
Ravindranath Konuru

Department of Computer Science and Engineering
Oregon Graduate Institute of Science & Technology

January 1992

This document is part of a series of reports describing the design decisions made in porting the Chorus Operating System to the Hewlett-Packard 9000 Series 800 workstation. This document presents an overview of the project, and outlines the other reports in the series and the relationships between them.

This research was supported by the Hewlett-Packard Company, Chorus Systèmes, and the Oregon Advanced Computing Institute (OACIS).

1 Introduction

This document is part of a series of reports describing the rationale, goals, technical and organizational issues that arose in porting the Chorus operating system to the Hewlett-Packard 9000/834 workstation (the 9000/834 workstation uses the Precision Architecture RISC 1.0 (PA-RISC) processor). This document presents an overview of the project and outlines the ther reports that are available in the series. The series of reports is intended for people interested in porting Chorus or interested in gaining a better understanding of the interaction between the PA-RISC architecture and the machine dependent (and independent) layers of a micro-kernel operating system.

The PA-Chorus project began in October 1990 with several goals. The members of the project had not previously performed an operating system port and therefore, one of the main objectives in undertaking the port was educational: we wanted to gain some general experience with low-level operating system code, and modern architectures. More specific goals of the port were to understand the interaction between micro-kernel operating system design and the HP PA-RISC architecture, and as a side-effect, to produce the infrastructure necessary to support distributed operating system research at Oregon Graduate Institute. An additional goal was to be able to offer a stable PA-RISC version of the Chorus kernel to other operating system researchers. However, the availability of the PA-Chorus kernel is largely in the hands of Chorus Systèmes and Hewlett-Packard since PA-Chorus contains a considerable amount of code that is owned by one or other of these parties. It is important to note that we never had the goal of producing a production-quality port of Chorus/MiX, or of achieving binary compatibility with HP-UX or other versions of Unix.

Funding for the project was provided through Hewlett-Packard's New Wave Computing University Grants Program, with additional support provided by Chorus Systèmes and Oregon Advanced Computing Institute (OACIS).

This document contains an overview of both the PA-RISC and Chorus, discusses the organization of the project, and outlines the content of, and relationship between, the other documents in the series.

2 PA-RISC Overview

The HP Precision Architecture program is the largest system development program ever undertaken by Hewlett-Packard Company. The Precision Architecture RISC (PA-RISC) was designed to serve as a common foundation for HP's computer systems and to excel in all of HP's computer markets. This section outlines the salient features of PA-RISC insofar as they relate to operating system design. More complete details of the architecture can be found in [10, 12, 8, 11, 6].

The memory hierarchy of the PA-RISC consists of registers (32 general-purpose, 25 control and 8 space registers), an architecturally visible cache (i.e., there are instructions for cache management), main memory, and an I/O system. The kernel and all processes share a global 64 bit virtual address space. Protection is supported at the granularity of pages using the following mechanisms: access rights (to specify the type of access); access identifiers (to restrict access to specific processes); and four privilege levels. Transfers between privilege levels are supported efficiently using a mechanism

that avoids trap handling overheads. A translation lookaside buffer (TLB) is used to support virtual address translation.

A prominent feature of the PA-RISC is that its cache is virtually addressed. Virtually addressed caches allow TLB and cache lookup to be performed in parallel. This may reduce cache access times which, in turn, can decrease the CPU cycle time and hence increase the speed of the processor. The cache is used both in virtual and physical addressing mode. In virtual addressing mode the cache is given (part of) a virtual address. In physical addressing mode the physical address is used to determine the index into the cache. Virtually addressed caches pose significant problems for operating systems that map physical pages to more than one virtual address at the same time. We will refer to this behavior as aliasing. The effects of aliasing are discussed in detail in some of the other reports in the series (see section 5).

In order to support a large, sparsely populated address space the PA-RISC architecture recommends using an inverted page table. Instead of maintaining an entry for every virtual page in the address space, inverted page tables maintain an entry for every physical page. This approach potentially reduces the size and complexity of page-table data structures, particularly if operating systems allow non-resident page information to be supported at a granularity coarser than pages.

Several features of the PA-RISC architecture are characteristic of prominent recent trends in computer architecture, particularly the move towards larger address spaces and virtually addressed caches. Consequently, many of the observations we made in porting Chorus to the PA-RISC have broader relevance.

Further details of the PA-RISC architecture and its interaction with operating system design (specifically Chorus) will be presented in the other reports in this series (see section 5).

3 Chorus Overview

Chorus is a micro-kernel-based distributed operating system architecture. The Chorus architecture consists of a minimal kernel (called the nucleus) that provides the mechanisms upon which to build higher level operating systems (called sub-systems). One of the most important sub-systems currently implemented above the Chorus nucleus is a Unix System V compatible sub-system called MiX. In this project we ported version 3.3 of the Chorus nucleus and version 3.2 of MiX to the HP PA-RISC. The current reports in this series cover only the Chorus kernel. However, in this section we outline the overall architecture of the Chorus/MiX system and discuss some of the salient features of its design. Further information on Chorus/MiX can be found in [3, 4, 1, 2, 5][1].

The Chorus nucleus has four key components: a communication manager, a virtual memory manager, a supervisor, and a multi-tasking executive. Each of these is discussed, in turn, in the following paragraphs.

The communication manager provides support for both local and global interprocess communication using the Chorus abstractions of *ports*, *port groups*, and *messages*. Messages in Chorus

[1] Chorus Systèmes technical reports are available via anonymous ftp from cse.ogi.edu (129.95.40.2) in directory pub/chorus-reports

are untyped and uninterpreted by the nucleus. Messages are also fixed in size. Chorus implements the transfer of virtual memory regions in the virtual memory manager, independently of the communication manager. In other words, unlike Mach, Chorus separates the issues of IPC and virtual memory. The communication manager consists entirely of portable code.

The virtual memory manager manages the memory local to a site (one Chorus nucleus runs per site). It provides the abstraction of a *context*: an address space which is divided into a number of non-overlapping *regions*. Regions define the valid portions of a context. The abstraction Chorus provides for a secondary storage object is called a *segment*. Regions are generally mapped to segments, and segments are managed outside the nucleus by segment *mappers*. The virtual memory system consists of a portable layer and a machine-dependent layer. The machine-dependent layer presents a generic memory management interface (i.e., a machine independent interface) to the portable layer. Many of the problems we faced in porting Chorus to the PA-RISC arose because of conflicts between the PA-RISC architecture and the definition of this interface (see [7] for details).

The supervisor is a dispatcher for external events such as interrupts and traps. Chorus allows system actors (an actor is the unit of resource allocation in Chorus, analogous to a task in Mach) to connect handlers to specific interrupts and to connect arrays of routines to trap invocations. Subsystems can use this facility to provide a flexible and efficient system call interface (as opposed to implementing sub-system calls using Chorus IPC, for example). Since the supervisor deals directly with the hardware (i.e., when handling traps and interrupts), and must manage processor-specific thread context information, much of the supervisor code is machine dependent (see [9] for details).

Finally, the multi-tasking executive provides priority-based preemptive scheduling and fine-grain synchronization. The multi-tasking executive is mostly portable.

The MiX 3.2 sub-system is a modular multi-server implementation of Unix System V release 3.2. MiX consists of a set of cooperating system servers, each dedicated to managing a particular type of resource, and running above the Chorus nucleus. The key system servers are the Process Manager, the File Manager, the Device Manager, the Pipe Manager, and the Socket Manager. The Process Manager provides a Unix system call interface by connecting its routines to traps, and hence must run in system mode (in the same address space as the kernel). Normally, the File and Device Managers also run in system mode in order to be able to execute privileged I/O instructions (the MiX 3.2 version of the File Manager contains the disk driver). Other servers run in user mode. Neither the Process Manager, nor the File Manager are completely portable. In other words, we had to do some work, mostly in the Process Manager, to port them to the PA-RISC.

The various components of the Chorus nucleus are discussed in more detail in the other documents in this series (see section 5 of this document) and in [3, 4, 1, 2, 5].

4 Porting Strategy and Organization

There were four members of the project team: two graduate students (Jon Inouye and Ravindranath Konuru), a systems administrator (Marion Hakanson), and a faculty member (Jonathan Walpole). The division of responsibilities was roughly as follows:

- Jonathan Walpole: Overall project management.
- Jon Inouye: porting the Chorus virtual memory system.
- Ravindranath Konuru: Porting the Chorus supervisor.
- Marion Hakanson: Remaining tasks and system administration.

None of the team members had performed an operating system port before, neither did they have any experience working with the PA-RISC or Chorus. Therefore, the early phases of the project involved a substantial amount of time for familiarization. Existing code was studied and reused wherever possible. In particular, we reused machine-dependent code from the PA-RISC versions of HP-UX, and Tut (a port of Mach 2.0 to the PA-RISC performed at Hewlett-Packard Laboratories), and machine independent Chorus code. We also had access to machine-dependent Chorus code for several other architectures (Motorola 88000, 68020, and Intel 386).

The project involved several identifiable phases. In the first phase we concentrated on familiarizing ourselves with the PA-RISC using high-level architectural documents from HP, and we studied the internals of Chorus by reading documentation and attending a course organized by Chorus Systèmes.

In the second phase of the project we started to reuse low-level boot code from HP-UX in order to build a bootable image that could perform primitive output, and eventually input, to the console. In particular we used the initial system loader ISL, and BOOT loader from HP-UX, and some low-level hardware initialization code from the Tut project. We also experimented with the *btChorus* utility in Chorus in order to build a single bootable image from several executable files. This phase of the project was considerably more time-consuming than we anticipated. There is no doubt that reusing existing code simplified our task considerably. However, we frequently spent large amounts of time struggling with problems that would have been trivial to solve had we had sufficient knowledge of the PA-RISC and the code we were trying to reuse. A brief visit from HP Labs employee, Bart Sears, one of the original Tut project members, proved extremely useful. However, even closer interaction with HP engineers could certainly have saved us much time and frustration.

The console input and output functionality in our intial standalone bootable image was supported using the PA-RISC's PDC (processor dependent code) and IODC (I/O dependent code) routines. These are ROM-based, software-callable routines resident on the processor and I/O modules respectively. The significance of these routines is that they provide primitive functionality for booting, initialization, and configuration. In particular, they allow console I/O to be implemented without requiring interrupts or device driver support. Had we obtained the documentation about these routines earlier in the project we could have saved much time and frustration.

Another task performed early on in the project involved writing utility routines for byte copying and zeroing pages.

The Chorus specific aspects of the port were divided broadly into two parts (*virtual memory* and *supervisor*), and each part was assigned to a separate individual. Jon Inouye ported the virtual memory system, which involved writing the MMU hardware initialization code, designing new

MMU classes and virtual memory-related data structures, writing code to handle virtual memory traps (such as TLB misses and page faults), defining the Chorus memory map, and performing the virtual memory aspects of Chorus initialization.

Ravindranath Konuru ported the supervisor, which involved writing code to manage thread contexts, perform interrupt, trap and exception handling, perform kernel initialization, define and manage thread contexts, manage simple devices (timer and console), implement a low-level debugger, and support system call interfaces for user and supervisor actors. Chorus kernel initialization requires virtual memory support, but does not initially require interrupt support. Therefore, we started out by implementing a kernel that did not support interrupts, and then introduced and tested interrupt support under the full kernel. An alternative approach would have been to incrementally test interrupt handling using the earlier standalone boot code. However, our schedule made the former approach more appropriate (i.e., because we didn't have the interrupt handling code written as early as we originally planned).

The testing and debugging phases of the project involved studying, porting, installing, and running the Chorus kernel test suite (K-tests). These programs test the full functionality of the kernel in both single pass and continuous modes. One problem we encountered at this stage was that we didn't have a machine on which to run a stable Chorus system. This would have made it much easier to understand the behavior of the test programs in an error-free environment. In retrospect, the lack of such a machine, combined with a lack of adequate documentation on the k-test source code and build environment, was a time-consuming mistake.

From a managerial standpoint we grossly underestimated the time and effort required in two key areas of the project. First, we didn't take into account the large overhead of administrative tasks such as setting up new machines, organizing source code (source tree layout) and build environment, installing new tools, providing utility programs for kernel I/O and debugging etc. Secondly, we didn't expect to be stalled on so many low level details in the early boot stages of the port. We would strongly advise anyone considering a similar venture to ensure that they have easy access to someone with extensive knowledge of the underlying architecture. Such a person should be prepared to answer questions potentially on a daily basis.

5 Outline of Other Reports in the Series

There are a total of six reports in this series, each one describing different aspects of the project. The report numbers and titles are as follows:

- "Porting Chorus to the PA-RISC: Project Overview," *OGI Technical Report No. CSE-92-003.*
- "Porting Chorus to the PA-RISC: Booting" *OGI Technical Report No. CSE-92-004.*
- "Porting Chorus to the PA-RISC: Virtual Memory Manager" *OGI Technical Report No. CSE-92-005.*
- "Porting the Chorus *Supervisor* and Related Low-Level Functions to the PA-RISC" *OGI Technical Report No. CSE-92-006.*

- "Porting Chorus to the PA-RISC: Building, Debugging, Testing and Validation" *OGI Technical Report No. CSE-92-007.*
- "Porting Chorus to the PA-RISC: Overall Evaluation" *OGI Technical Report No. CSE-92-008.*

The first report (this one) presents an overview of the project and provides perspective for the other reports. This report is intended to be read in conjunction with one or more of the other reports in order to gain any detailed information about the project. Report 2 describes the approach taken to booting PA-Chorus, including a discussion of the various options that were open to us, our reuse of low-level code from Tut, virtual memory initialization, and the Chorus boot and initialization procedure. Report 3 discusses the Chorus and PA-RISC virtual memory systems. It covers the major design decisions and presents details of the design and implementation of the Chorus machine-dependent classes for the PA-RISC. Report 4 covers a host of issues relating to the Chorus supervisor. Key issues include: an overview of the supervisor design, interrupt, trap, and exception handling on the PA-RISC, system call handling, the reuse of low-level handler code from Tut, thread context structure, context switching, and memory management trap handling. Report 5 covers the approaches we took to building, debugging, testing and validating the PA-Chorus kernel. Finally, report 6 presents an overall evaluation of the matches and mismatches between Chorus and the PA-RISC.

6 Acknowledgements

Several people contributed their time and energy to help us complete this project. We are particularly grateful to Jean-Jacqués Germond, Frédéric Herrmann, Vadim Abrossimov and Olivier Giffard of Chorus Systèmes, and Bart Sears, Jean Gascon, Ahmed Ezzat, Howard Stateman, and Charlie Roberts of Hewlett-Packard Company.

References

[1] Vadim Abrossimov, Marc Rozier, and Michel Gien. Virtual Memory Management in Chorus. In *Proceedings of Progress in Distributed Operating Systems and Distributed Systems Management.* Springer Verlag, April 1989. Also published as technical report CS/TR-89-30.

[2] Vadim Abrossimov, Marc Rozier, and Marc Shapiro. Generic Virtual Memory Management for Operating System Kernels. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, December 3-6 1989. Also published as technical report CS/TR-89-18.

[3] François Armand, Michel Gien, Frédéric Herrmann, and Marc Rozier. Revolution 89 or "Distributing UNIX Brings it Back to its Original Virtues". In *Proceedings of the Workshop on Experiences with Building Distributed and Multiprocessor Systems*, October 5-6 1989. Also published as technical report CS/TR-89-36.

[4] François Armand, Frédéric Herrmann, Jim Lipkis, and Marc Rozier. Multi-threaded Processes in CHORUS/MIX. In *Proceedings of EUUG Spring'90 Conference*, pages 1–13, April 23-27 1989. Also published as technical report CS/TR-89-37.

[5] CHORUS Kernel v3.3 Implementation Guide. Technical Report CS/TR-90-71, Chorus Systèmes, 1991.

[6] Hewlett-Packard. *Precision Architecture and Instruction Set Reference Manual*, third edition, April 1989.

[7] Jon Inouye, Marion Hakanson, Ravindranath Konuru, and Jonathan Walpole. Porting Chorus to the PA-RISC: Virtual Memory Manager. Technical Report CSE-92-5, Oregon Graduate Institute, January 1992.

[8] David V. James, Stephen G. Burger, and Robert D. Odineal. Hewlett-Packard Precision Architecture: The Input/Output System. *Hewlett-Packard Journal*, 37(8):23–30, August 1986.

[9] Ravindranath Konuru, Marion Hakanson, Jon Inouye, and Jonathan Walpole. Porting the Chorus Supervisor and Related Low-Level Functions to the PA-RISC. Technical Report CSE-92-6, Oregon Graduate Institute, January 1992.

[10] Ruby B. Lee. Precision Architecture. *IEEE Computer*, 22(1):78–91, January 1989.

[11] Joseph A. Lukes. HP Precision Architecture Performance Analysis. *Hewlett-Packard Journal*, 37(8):30–39, August 1986.

[12] Michael J. Mahon, Ruby Bei-Loh Lee, Terrence C. Miller, Jerome C. Huck, and William R. Bryg. Hewlett-Packard Precision Architecture: The Processor. *Hewlett-Packard Journal*, 37(8):4–22, August 1986.