

## EXPERIENCE PORTING THE HARMONY OPERATING SYSTEM

J. James, K. Rowe, L. Gray, B. Vishnubhatla, C.F. Wan, M. Wilson

ARTT Project  
Systems & Computer Engineering  
Carleton University, Ottawa, Canada

DY4 Systems  
888 Lady E-len Place  
Ottawa, Canada

### ABSTRACT

This paper discusses an industrial and academic research groups' experience resulting from porting a real-time multiprocessing operating system from one host-target environment to a new host and target environment.

Keywords: portable, real-time, multiprocessor

### 1. Introduction

Current VLSI technology has created near exponential decreases in cost/performance for processors, memory, and peripheral support circuits. The manufacturers of single board computers have been one of the first groups to exploit these advances in the production of new inexpensive single board computers.

These single board computers are supported by a wide variety of I/O devices and use a number of industry standard busses; for example, VME-bus, MULTIBUS II and Nu-bus. All of these bus structures support multiprocessing, and with the low cost of microprocessors, system designers are beginning to capitalize on this feature.

Effective utilization of a shared bus multiprocessor system is not without its problems. Systems designers require inexpensive, reliable solutions to real-time problems and flexible solutions to minimize life cycle costs while maintaining real-time performance. The current solutions, which use a collection of uniprocessor kernels or a centralized multiprocessing kernel<sup>1,2</sup>, fail to meet the needs of the system designer. A real-time multiprocessing operating system which is portable to a wide variety of hardware is required. The Harmony Operating System provides high level I/O support, real-time performance, portability and transparent multiprocessing making it ideally suited to this environment.

This paper presents a case study on the porting of the Harmony Operating System to a VME based multiprocessor system. This paper focuses on the specific issues involved in performing a port of this nature and the problems of portability for the Harmony Operating System.

The porting problem is determined largely by what is being ported. A very specific application such as a memory diagnostic can be ported to a new system easily. Such a test can be implemented

directly in a high level systems programming language. From this, one can conclude inductively, that a port depends upon the functionality of the ported software and hardware constraints. Another consideration introduced by the Harmony Operating System is the host development environment. In an open system one must also consider users' development tools, because it should not be necessary to change tool sets every time a new project is started. Therefore, before describing the porting methodology, the original or status quo host/target system must be described or else the significance of the port will be missed.

This paper begins with a discussion of the Harmony Operating System, its current features, and its hardware requirements. Next, several sections are devoted to: the porting approach used by the authors, the testbed environments, hardware modifications, software modifications, and organizational problems. The final section summarizes the findings of the group.

### 2. A Description of Harmony

Harmony is a real-time, multitasking, multiprocessing, operating system for embedded computer applications. It is implemented in the "C" programming language with a small amount of assembler; therefore, harmony is portable across development environments, and different target hardware.

Harmony reflects the strong influence that the THOTH operating system<sup>3</sup> had on its beginnings. Harmony was developed over the past three years at the National Research Council in Ottawa, Canada. It is continuing to evolve today.

The following sections discuss the overall features and requirements of the Harmony Operating System under the subheadings: hardware independence, portability, real-time features, open system features, multiprocessor support, device support, harmony primitives, host file structures and software management, and tools.

#### 2.1 Hardware Independence

Harmony makes four key assumptions about the hardware and associated support.

- Linear addressing is provided.
- A test-and-set operation is available to

provide mutual exclusion for the multiprocessor.

- A processor must be able to interrupt any other processor including itself.
- The processor must support multiple interrupt levels\*

The addressing and memory considerations required to support the Harmony Operating System centre around the concept of a linear address space. This means that all memory and I/O in a system can be uniquely addressed by any processor in the system. One possible hardware solution is global memory; however, this leads to bus saturation. A current promising solution to this problem is providing completely dual ported memory for each processor. This reduces the bus traffic drastically.

Linear addressing is required for several reasons. With a completely linear address space, references can be embedded directly in messages. During the message transfer process, a new copy of the message is made and the transfer can be implemented directly. The operating system data structures rely upon a linear address space. The operating system kernel is distributed, therefore each processor has its own set of kernel data structures. However, task state information must be accessible by all processors to support message passing and task management.

The interrupt support for harmony requires that each processor be capable of interrupting every processor in the system, including itself. The simplest method of providing this feature is through a set of I/O registers which are accessible to all by virtue of the linear address space. The interprocessor interrupts on the VME bus may be used in replacement of these locations if the specific hardware support is not available.

## 2.2 Real-Time Features

The Harmony Operating System views the concept of real-time as two distinct issues. First, the programs must interact with I/O devices which communicate with the real world. This means that asynchronous events must be handled by the system. Second, time is seen as a critical resource which must be managed<sup>4</sup>. This means that specified response times, periodic scheduling and volatile data storage times must be respected.

Harmony provides the following features to deal with these requirements:

- (a) Harmony provides a user defined number of static priority levels for tasks.
- (b) Tasks run on the processor until a "natural break"

-----  
\*This is not precisely correct; however, to provide reasonable real-time performance, this is a requirement.

\*A natural break is defined as preemption by a higher priority task due to an event, or by blocking.

- (c) One task may await for one interrupt using an await-interrupt call.
- (d) Volatile data can be saved during the initial interrupt response, and passed on to the waiting task via the await-interrupt message.
- (e) Periodic scheduling of a collection of tasks may be added via explicit inter-process communication.

Items (a) and (b) ensure that time responses may be guaranteed by providing deterministic scheduling. Items (c) and (d) provide appropriate interrupt service saving any volatile data. The last term provides periodic scheduling to applications.

## 2.3 Portability

Portability in the harmony context means portability across development environments and target hardware. At the present time, the critical host support requirements are a "C" language cross compiler, a compatible cross assembler, and a compatible linker which supports address relocation.

Portability from the target hardware point of view means not only "C" and assembler support for the target processor, but must be expanded to include the following items:

- a minimal amount of assembly language conversion
- a reasonable method to specify stack frame initialization and/or other processor specific functions
- support for a variety of peripheral chips
- system bus independence
- host target interface for downloading, testing, etc.

## 2.4 Open System

The Harmony Operating System was developed with the intent that it be an "Open System". The following describes briefly what an Open System means in this context.

An Open System has all of the aspects of portability that we have already defined. The initial Harmony system used a Motorola 68000 processor. This document describes a port to a Motorola 68010 multiprocessor system. At the time this paper is written ports to systems using the Motorola 68020 and National Semiconductor 32016 are planned.

Standard host development tools, as already stated, must also be considered. The only two constraints limiting selection of a host are:

- (a) The host must support the C language compiler compatible with C as defined by Kernighan and Ritchie<sup>5</sup>.

- (b) The file structure supported by the operating system on the host should be tree structured as in the UNIX\* operating system.

Constraint (a) is mandatory but not too restrictive. Constraint (b) is not mandatory because developers can do whatever they wish to the Harmony files. However, the status quo file structure used to support Harmony has already been proven in a research and development environment and helps a developer to use and maintain Harmony<sup>6</sup>. Table 1 lists development systems that presently host the Harmony Operating System.

Table 1  
Host Systems With Harmony  
Operating System Supported

hardware	Operating System	C language
VAX*	VMS*	Whitesmiths C
VAX*	UNIX** Berkley 4.2	Portable UNIX** C
WICAT	MCS	Portable UNIX** C
DSM-6816	UNIX** Uniplus*** System III	Portable UNIX** C

\*VAX and VMS are trademarks of Digital Equipment Corp.  
\*\*UNIX is a trademark of AT&T Bell Labs  
\*\*\*Uniplus+ is a trademark of Unisoft Corp.

An Open System must support many different peripheral devices as well as different processors. To meet this end, the Harmony kernel is device independent. All device handling (handlers are called servers) is defined completely by applications software.

The systems programmer is not constrained to write applications using C. In principal applications can be written using Pascal and FORTRAN.

## 2.5 Support for Multiprocessing

A feature of the Harmony Operating System that makes it different from conventional microprocessor operating systems is the capability for multiprocessing. This capability is supported by specific features of Harmony and by the hardware to which it has been ported.

Externally, that is to the application, Harmony removes hardware considerations with the process-id abstraction and inter-process communication. Recall, Harmony is a message passing operating system. Using this type of inter-process communication, an application process never needs to know where a process resides; the name of a process or process-id is all that is required. Therefore, with hardware considerations removed, multiprocessing can be added with appropriate hardware and some simple kernel features.

\*UNIX is a trademark of AT&T Bell Labs

Internally Harmony multiprocessing is supported by the inter-processor interrupt, an interconnection network, a hardware address, a mailbox and shared memory. The inter-processor interrupt allows one processor to interrupt another so that a message can be exchanged. An interconnection network provides an access channel between processors and from processors to shared memory. At present the Harmony Operating System is implemented on an interconnection network type more commonly known as a Time Shared bus<sup>7</sup>. (Two examples of this type of interconnection network that Harmony has been ported to are Multibus<sup>4,8</sup> and VME-bus<sup>9</sup>. The port described in this paper is to a VME-bus system.) A unique hardware address range coded in the process-id maps a process to one specific processor. A unique mailbox bound to every processor provides a place to leave a message address. Dual port memory is used to store messages so one processor can access a message from another processor.

## 2.6 Device Support

A major feature of Harmony is the provision for logical device support, which is independent of the underlying target hardware. These "servers" provide high level I/O abstractions to eliminate detailed understanding of hardware devices and "rebuilding the wheel" each time a new application is attempted<sup>10</sup>. Currently these servers provide a clock abstraction, a stream I/O service similar to UNIX file I/O for physical devices, and a file server.

The special server features are largely the result of the fact that the kernel is independent of these servers, and they are simply a collection of tasks. Therefore, the users may provide their own servers for specialized applications. Furthermore, servers can be passed configuration records at initialization time, which can dynamically change the features of a given generic server type.

## 2.7 Harmony Primitives

Figure VI lists the primitives provided by the Harmony Operating System Kernel. Details on these primitives are in reference 1.

## 2.8 Host System File Structure and Software Management

Probably the most important part of the Harmony Operating System is the file structure used to support it. The file structure defines all of Harmony. It contains all Harmony Operating System source and files to build executable images. A tree like file structure is used. Figure I illustrates an example of the higher part of the tree. Figure II illustrates an example of the lower part of the tree.

The file structure used was first applied in the Thoth Portable Operating Systems Project at the University of Waterloo<sup>3</sup>. A tool for managing a large software project it is very different than

```

Process Management      unsigned _Create (task_index)
                        unsigned task_index
                        unsigned _Server_create (task_index,
                        unsigned task_index,
                        char      *init_msg)
                        unsigned _My_id ( )
                        unsigned _Father_id ( )
                        _Destroy (id)
                        unsigned id
                        _Suicide ( )
                        unsigned _Task_error_code ( )

Process Communication   unsigned _Send (rqst_msg, rply_msg, id)
                        char      rqst_msg [ ], rply_msg [ ]
                        unsigned id
                        unsigned _Receive (rqst_msg, id)
                        char      rqst_msg [ ]
                        unsigned id
                        unsigned _Reply (rply_msg, id)
                        char      rply_msg [ ]
                        unsigned id

Input/Output           struct UCB * _Open (name, mode)
                        char      *name
                        short int mode
                        _Close (ucb)
                        struct OCB *ucb
                        struct UCB * _Selectinput (ucb)
                        struct UCB *ucb
                        struct UCB * _Selectoutput (ucb)
                        struct UCB *ucb
                        char_Get ( )
                        _Put (byte)
                        char byte
                        _Flush ( )
                        _Unget ( )
                        _Getnum ( )
                        _Putdec (n)
                        _Puthex (n)
                        int n
                        _Putstr (s)
                        char *s
                        _Printf (fmt, x)
                        char *fmt
                        int x

Pool Management        char * _Getvec (size)
                        unsigned size
                        _Freevec (block)
                        char *block
                        _Trimvec (block, size)
                        char * block
                        unsigned size

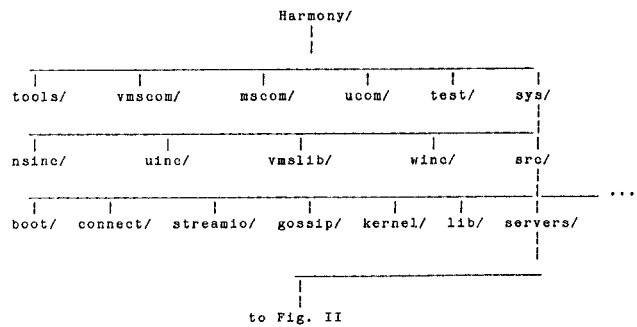
Device Management      _Await_interrupt
                        (interrupt_id, rply_msg)
                        unsigned interrupt_id
                        char rply_msg [ ]

```

Figure VI Harmony Operating System Kernel Primitives

conventional software management tools like SCCS<sup>6</sup>. Individual software modules can be kept small. In most cases there is one function per file. As illustrated in Figures I and II, the system is highly structured. The structuring promotes modularity. Individual modules are easier to maintain. However, at present there are no tools to maintain the structure.

Presently the system contains about 1000 files which includes: the kernel, all devices, tools, Motorola 68000 port, Motorola 68010 port, National Semiconductor 32016 port.



/indicates a directory

Figure I  
File Tree From Root of Harmony to Servers Directory

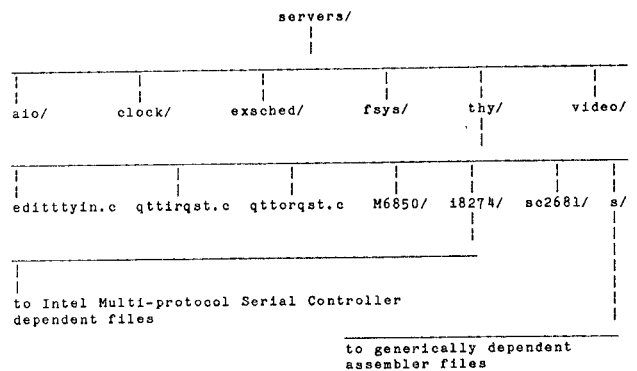


Figure II  
File Tree From Servers Directory  
to The Device Dependent Directories

## 2.9 Tools

The Harmony Operating System has specific tools that have been developed to aid the user. When it is ported the tools must also be ported. Some of the tools are: Listing, Bound, Examine, and Debug.

The Listing tool produces a listing, with table of contents and a banner from an include file containing quoted names of the files listed. Usually the same include file is used to compile an executable unit of a code.

The Bound tool calculates the stack size required for a process. Bound does this by disassembling the executable image of the process. It is interactive, prompting for assistance when recursion is encountered or the stack pointer is indirectly loaded.

The Examine tool allows interactive examination including disassembly and patching of the executable image. It is not a run time tool, but is used on the image in the host.

The Debug tool provides a simple debugging capability and interpretation of kernel data structures. The Debug tool is invoked explicitly in the source and when an exception handler is invoked.

### 3. Starting Point

This section describes the conditions and limitations dictating the task of porting a prototype development.

#### 3.1 Documentation

A substantial software system was ported. The software was in a pre-release state and had not been documented thoroughly. There were two documents. One was an overview of: terminology, the server concept, kernel primitives, interrupt handling, I/O and tools<sup>4</sup>. The second document was a note describing rationale for the Harmony file structure<sup>6</sup>. Documentation that would have been helpful was a manual describing the mechanics of designing a server and a description emphasizing hardware dependent areas affected when a system is ported. Source code documentation, file system documentation, and details of the Harmony Operating System internal features were also lacking.

#### 3.2 Hardware

The Harmony Operating System being ported was implemented on the Chorus Multiprocessor. This multiprocessor uses 3 Omnibyte OB68K1A Single Board Computers (SBC). These SBCs use the Motorola 68000 microprocessor and are Multibus compatible. The interprocessor interrupt is implemented with a custom made Multibus card and additional backplane connections. Basically, this card is an array of writable registers connected to bus interrupt lines. Each interrupt line is connected to one SBC and each register controls one interrupt line.

The target system multiprocessor consists of 4 DY-4 DVME-102 Single Board Computers. These SBCs use the Motorola 68010 microprocessor, were configured to support a linear address space, and are VME-bus compatible. The interprocessor interrupt was implemented using the control status register available on each SBC. An SBC is interrupted whenever this register is written.

#### 3.3 Physical Proximity

As stated above, there was little supporting documentation. This problem was compounded by two factors. First, the original developers were at a separate location, and as a result, only telephone support was available on an instantaneous basis. Also, the porting team was located at two different sites making internal communication difficult. Our plans had to consider these problems in order to avoid duplication of effort.

#### 3.4 Development Environments

The National Research Council used two development environments: VAX running VMS and a

WICAT running MCS. The development environments ported to are both UNIX environments: a VAX running Berkley 4.2 and a DSM-6816 running Uniplus+\* System III. Table 1 lists these environments and the system development languages available with each system.

The basic problems encountered when porting from one host to another were: file naming conventions and incompatible libraries.

### 4. Porting Approach

Two distinct objectives characterized the port:

- (a) One objective was longer term. It required that a complete Harmony system be ported along with all servers and tools. This port would be capable of becoming a marketable product on a DY4 DVME-102 multiprocessor in a six month time frame.
- (b) The second objective was both short and long term. The short term aspect reflected the fact that a working system was needed in the immediate future to generate credibility and to support research efforts. The longer term objective required a research system with sufficient support and understanding to evaluate the feature set of the operating system.

#### 4.1 Project Organization

Given the constraints imposed, the project organization consisted of two separate teams each with different objectives and responsibilities.

Team A: This team had three part-time members and would attempt an immediate MC68010 port. It would concentrate on understanding the system and finding the problems.

Team B: This team had three full-time members and would attempt a longer term production version of the system. Its focus would be on documentation, and a complete, reliable, robust implementation.

#### 4.2 Team A Activity Breakdown

- (a) The file system was reduced to a minimal, manageable set required for this rapid port. Source code control and project management support were introduced.
- (b) Basic documentation of the system was started.
- (c) The MC68010 uniprocessor port was completed without stream I/O.

---

\*Uniplus+ is a trademark of Unisoft Corp.

- (d) The MC68010 multiprocessor port was completed without stream I/O.
- (e) The I/O was added (using input from Team B).
- (f) The system was used until the production version became available, at which point the research version was discarded.

**4.3 Team B Activity Breakdown**

DY-4 Systems adopted a 3 phased porting methodology. The phases were:

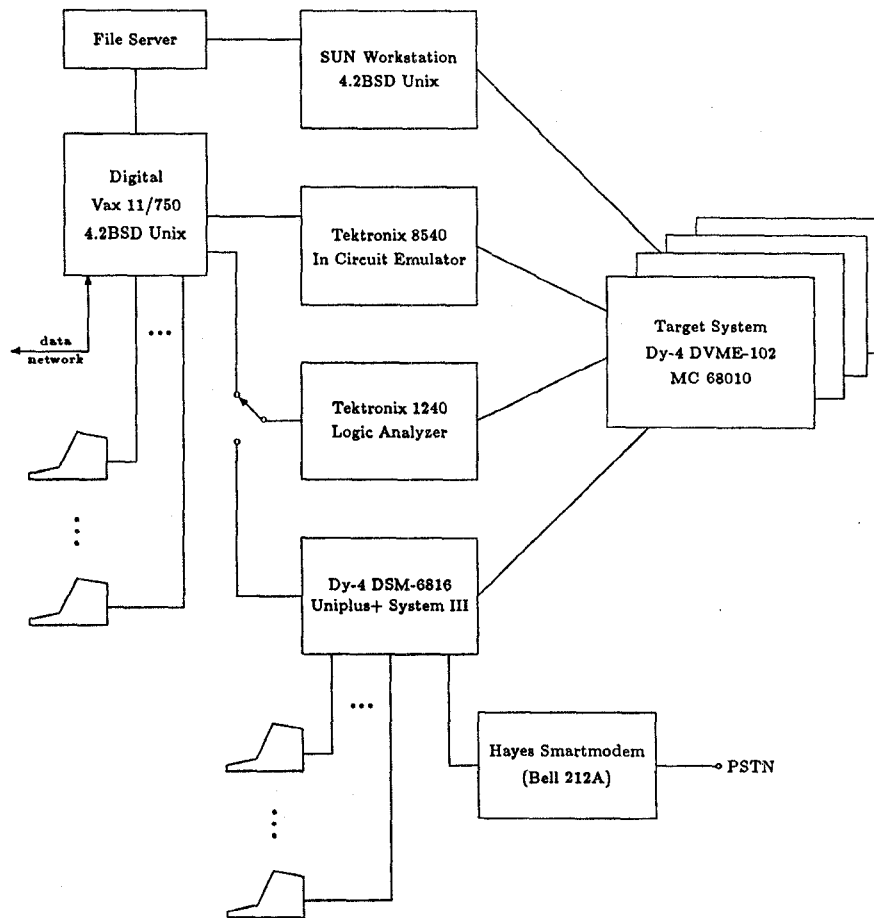
- (a) single processor 68000 port
- (b) single processor 68010 port
- (c) multi-processor 68010 port.

\*SUN is a trademark of Sun Microsystems  
 \*\*Vax is a trademark of Digital Equipment.

**4.4 Team A Environment**

Figure VII illustrates the hardware architecture of the laboratory environment. The main features are as follows:

- Both the SUN\* workstations and the VAX\*\* run 4.2 BSD. Transparent login, remote csh, and file transfer are supported.
- The DY4 DSM-6816 runs UNIX System III.
- Kermit and UUCP are used for file transfer between the Vax and the DSM.
- The Tektronix 8540 ICE has extensive support software on the Vax.
- The Tektronix ICE was used to download to target processor during development.
- Source code control and project management software were available on the VAX.
- The Tektronix logic analyzer with MC68010 support.



**Fig. 7 The ARTT Laboratory Hardware**

**4.5 Team B Environment**

The host development system used at DY-4 was a DY-4 minicomputer called the DSM-6816. This system runs Unisoft Uniplus+ UNIX System III. Specifically the DSM-6816 uses VME-bus architecture and consists of a DVME-102 processor card and a DVME-712 Intelligent Disk Controller. Figure VIII illustrates the environment.

The environment was augmented by an Applied Microsystems ES 1800 Satellite Emulator. This emulator is very easy to use and is easily interfaced with the host.

A Hewlett-Packard 1630 logic analyser was also used. This type of tool is usually necessary for debugging the hardware systems software interface because it can be used to remove much of the guess work that results when a device handler does not work.

**5. Summary of Modifications**

This section is broken down into three sections: Team A, Team B, and hardware modifications. The hardware modifications were a joint effort, therefore not considered separately.

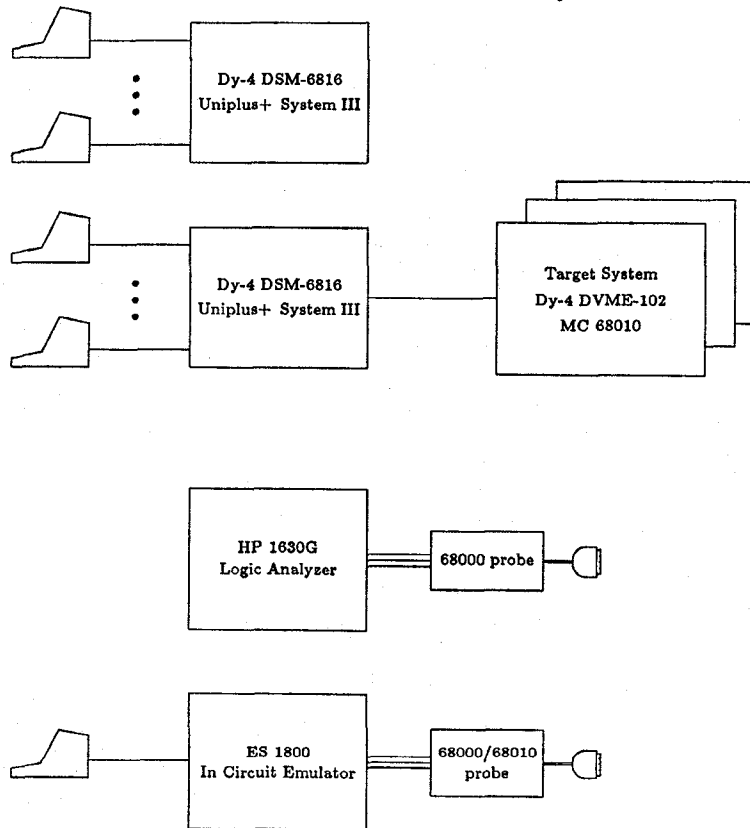
**5.1 Team A Modifications**

- (a) The initial preparation for the MC68010 port required a significant portion of the total effort expended. The major reason for this was the variety of new information.
  - (i) An understanding of Harmony internals was developed from the undocumented source code.
  - (ii) New programming environments existed on both hosts.
  - (iii) An unfamiliar target system was used.
  - (iv) All logic instruments were new and required a detailed understanding.

Given these constraints, a great deal of time was used for learning, and then a "learn as you go" approach was used.

The other issues involved in preparation included the following points:

- (i) The Harmony source code was moved to the host system into a simplified file system.



**Fig. 8 The Dy-4 Software Development Environment**

- (ii) Basic documentation was completed on the source code.
  - (iii) A basic porting strategy was developed.
  - (iv) A source code control system was installed to ensure that a working version could be maintained.
  - (v) Project control software was added to facilitate listing source and making global modifications.
- (b) The efforts for a uniprocessor port on the existing DVME-102 card required more learning about the primitives and the environment. The basic steps were:
- (i) The stream I/O facility was removed from Harmony to simplify the port.
  - (ii) Harmony primitives were modified to suit the hardware.
  - (iii) The system was recompiled and linked. A number of inconsistencies were eliminated at this stage.
  - (iv) The primitives were debugged, largely using the ICE.
  - (v) Note that the Harmony tools were not ported or used during the debugging.
- (c) The basic steps followed were as follows:
- (i) Hardware testing
  - (ii) Modification of primitive routines
  - (iii) Testing of primitive routines on a uniprocessor
  - (iv) Multiprocessor testing and debugging.
- (d) The final step of adding the stream I/O support and the character server was not carried out. The two ports were operational essentially at the same time, and this step was no longer necessary.

## 5.2 Team B Modifications

### (a) Single Processor Motorola 68000 Modifications

A 68000 version was selected initially, because it required fewer machine dependent modifications. This port consisted of these subphases:

- acquire VMS\* version of Harmony from the National Research Council
- convert VMS\* pathnames to UNIX pathnames

---

\*VMS is a trademark of Digital Equipment Corporation

- convert assembler language files to format accepted by assembler on UNIX
- design new generic serial I/O server for serial I/O device on target SBC
- modify interprocessor communication to support new interprocessor interrupt mechanism
- design standalone DVME-102 test software to verify hardware configuration
- test and verify processor  $\bar{a}$
- retrofit processor  $\bar{a}$  with new PALs
- retest and reverify processor  $\bar{a}$
- design single processor application to test kernel and serial I/O server
- build and test Harmony Operating System and application
- modify listing tool to use UNIX library
- modify Bound and Examine tools for UNIX object file format
- build and test all tools.

### (b) Single Processor Motorola 68010 Modifications

In the second phase the first phase port was ported to a single processor Motorola 68010 system. At the same time the multiprocessor hardware configuration was completed.

This port consisted of these subphases:

- modify previous port for larger exception stack frames of Motorola 68010
- define and implement application level specification of user exception handlers
- complete and test PALs for up to 8 processors in the multiprocessor testbed
- define, implement and test multiprocessor enhancements to standard DVME-102 debug monitor
- implement and test port using same application program used in (a).

### (c) Multiprocessor Motorola 68010 Modifications

This phase completed the port of the Harmony Operating System. This port consisted of these subphases:

- modification of the file defining memory limits of each SBC.
- modification of data structure defining all SBC system control status registers

- define test program
- build, implement and test port.

### 5.3 Hardware Modifications for Multiprocessor Support

During the preparation of the uniprocessor versions by both groups, concurrent effort was expended to solve the problems relating to the multiprocessor hardware. This effort was undertaken jointly by both groups to provide a common hardware solution using DY4 DVME-102 processor cards with minimal hardware modifications.

The DY4 DVME-102 product has the following features

- 68000/68010 CPU @ 8/10 MHz
- optional 68451 mmu
- 256K to 1/M bytes RAM, completely dual ported
- 8274 serial I/O
- 3 timers
- DY4 Monitor
- VME interfaces with system controller functions
- extensive strapping options
- PAL based design allowing easy customization.

One major hardware requirement of Harmony is prioritized interrupt levels. This feature was provided by virtue of the MC68000 family.

Another major feature of the Harmony Operating System is linear addressing. The current 102 product's memory map is shown in Figure III. Modification of this addressing scheme required three new PALs per processor and special strapping of the boards. This produced the changes shown in Figure V and Figure VI.

The ability to interrupt any processor in the system is required for Harmony. On the current 102 board, two bus interrupts can be generated via the I/O control status register. It is possible to restrap the board to generate a local level five interrupt using these bits. This was done, and along with the addressing modifications, this produced the required interrupt locations. Note that these locations are offset from each other by judicious placement of the processor number in bits 17-19 of the I/O control status register address. This provides a simple implementation of interrupt location addressing for Harmony primitives.

The final feature required to support multiprocessing is a set of ROM monitors which will synchronize the system during initialization. It was decided to use the DY4 monitor for downloading/disassembly purposes, and modify it to provide the synchronization. The synchronization

### 68000/68010 PHYSICAL ADDRESS MAP

Address Range (Hex)	Type	Description
FFXX00 to FFXFFF	I/O	on-board peripherals
FF0000 to FFFFFF (other than on-board I/O)	I/O	VMEbus I/O access
FE0000-FEFFFF	Memory	boot PROM
100000 - FDFFFF	Memory	VMEbus memory
10000 - FFFFF	Memory	on-board RAM
0 - FFFF ('boot' not active)	Memory	on-board RAM
0 - FFFF ('boot' active)	Memory	boot PROM

Note: boot PROM capacity is 32K bytes although it occupies a 64K byte window

FIGURE III STANDARD DVME-102 MEMORY MAP

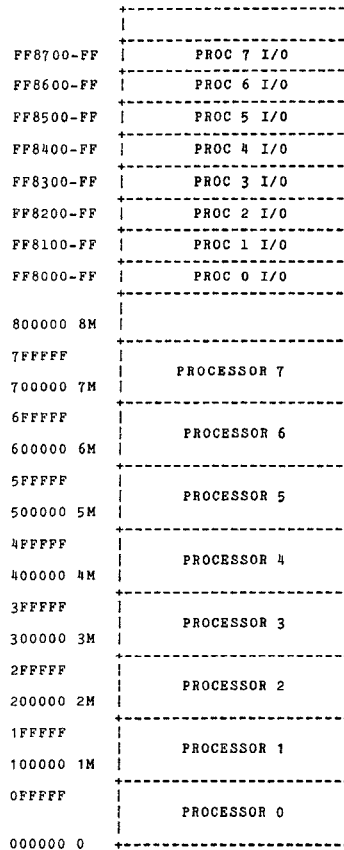


FIGURE IV VMEbus MEMORY MAP FOR FULLY CONFIGURED SYSTEM

of the system is processor dependent and the value of the processor number is required for initialization. This problem was solved by using a readable jumper block to provide the processor number. This allowed a complete set of processors to use identical boot ROMs because the processor simply checked its processor value and provided the appropriate synchronization.

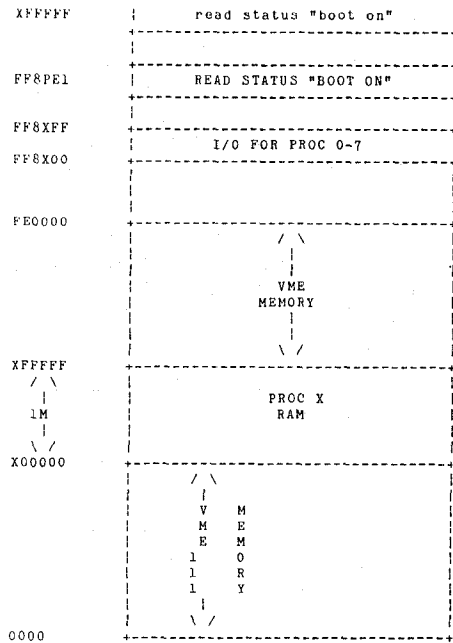


FIGURE V ON-BOARD MEMORY AND I/O MAP FOR PROC X, (X=1,7)

In summary, all four hardware requirements for Harmony were satisfied using the same off the shelf processor card. Three PALs and two ROMs were changed on each card.

## 6. Results

### 6.1. Team A Results

Overall, the porting effort of team A consumed 8.5 man months or approximately 182 days. A detailed breakdown is given in the following subsections.

#### (a) Initial Learning Phase

An estimated 5.5 man months was consumed during this phase. In general, estimation was difficult because accurate records were not kept and learning effort in some areas was spread over several projects. An estimated 2.5 man months could have been saved if proper documentation had been included with the source. Furthermore, another 2 man months could have been saved if one person had undertaken the project as opposed to three part time members.

#### (b) Uniprocessor Porting Results

The initial uniprocessor porting efforts started and consumed three-quarters of a man month before it was discovered that an ICE was essential. When efforts resumed, one part-time person completed the port in one man month.

Three major problems arose during the port. First, stack frame initialization proved troublesome. Second, interrupt modification was a problem. Third, some changes resulting from different exception stack frames were required. Consultation with the original designer quickly solved these problems; however, proper documentation would have eliminated one week of time (minimum) through more rapid comprehension of the problems.

The multiprocessor MC68010 port required hardware modifications (discussed in the next section) and changing priorities.

#### (c) Multiprocessor Port Results

Hardware modification definition required approximately one quarter of a man month for team A. The remainder of the multiprocessor port required 0.5 man months. The total effort to complete the port was 10% greater than estimated after completion of the uniprocessor port. Experience effects were significant in this estimate. Total effort was 0.75 man months.

#### (d) Stream I/O Results

Approximately 0.5 man months were wasted when duplicate serve porting efforts were started. This was a direct result of poor communication.

### 6.2 Team B Results

#### (a) Single Processor Motorola 68000 Analysis

The total time taken to complete this phase was 7 weeks. There were 3 people working full time on the project during this phase.

A total of 10 coding errors and 1 hardware error were uncovered and resolved. Only 5 of 35 days were spent testing and debugging the port. This is due largely to preparation and constant informal review. All software errors were a result of changes made to the operating system or due to the new serial I/O server.

#### (b) Single Processor Motorola 68010 Analysis

The total time required to complete this phase was again 7 weeks. One week was used to debug the multi-processor monitor enhancements. One day was used to debug and verify the port. There were 2 people dedicated to the port during this phase.

A total of 8 coding errors were resolved to debug the multiprocessor monitor; and 2 coding errors were resolved to debug this phase of the port. During this phase the monitor enhancements were not reviewed at the source level but the port modifications were.

(c) Multiprocessor Motorola 68010 Analysis

The total time taken to complete this phase was 4 days. During this phase there were 3 people dedicated to the port.

A total of 2 coding errors were resolved to debug this phase of the port. This phase was a lot shorter than the previous two phases because the modifications were minor. Building of the port required the recompilation of 2 kernel files and the test application.

## 7. Conclusions

- (a) An estimated 45% of the total porting effort could have been eliminated given adequate source documentation.
- (b) The total time spent by the two teams was approximately equal; however, team B accomplished considerably more. The reasons for this are as follows:
  - (i) Team B had experience in their environment while Team A did not. The group/personnel experience effect is one of the most significant [11], and this is reflected here.
  - (ii) Part-time dedication to the porting required more duplication of effort for Team A.
  - (iii) The results of Team A were used by Team B to eliminate bugs in their MC68010 uniprocessor and multiprocessor system.
- (c) The communication problems, sharing of information, and disperse location problems were minimized through autonomous tasks at the two locations. Co-location would have been the best solution; however, it was not possible. An autonomous tasks solution did work.
- (d) Experience effects were crucial in refining time estimates of porting the software.

- (e) Overall, the porting was technically easy. The main problem was developing a sufficient understanding of the software. Clearly, adequate documentation is the single most significant factor in developing this understanding and must be a central part of any portable system. This is supported by conclusion (a).
- (f) The involvement of many people on a less than full time basis proved a significant hinderance to the research group. Significant effort and particularly learning effort was duplicated. A full time effort could have reduced the effort by an additional 15% assuming adequate documentation.
- (g) Downloading at 9600 BPS was a significant hinderance.
- (h) The ICE was essential for debugging, and the host support for the ICE helped immeasurably.

## **ACKNOWLEDGEMENTS**

Funding for this work was available from National Research Council Canada contribution arrangement CA910-4-0036/670, a National Sciences and Engineering Research Council Grant, and Ottawa Carleton Research Institute.

## BIBLIOGRAPHY

- [1] "MTOS-68K User's Guide", Industrial Programming, Inc., Jericho, New York 11753, 1981.
- [2] P.K. Rowe, "Microprocessor Interprocess Communication Using Ada, Monitors, and VME", AIAA Computers in Aerospace IV Conference, Oct. 1983.
- [3] D.R. Cheriton, M.A. Malcolm, L.S. Melen, and G.R. Sayer, "Thoth, a Portable Real-Time Operating System", Communications of the ACM, Vol. 22, No. 2, February 1979.
- [4] W.M. Gentleman, "Using the Harmony Operating System", ERB-966, NRC No. 23030, National Research Council of Canada, Oct. 1984.
- [5] B.W. Kernighan, M. Ritchie, "The C Programming Language", Prentice-Hall Software Series, 1978.
- [6] T.A. Cargill, "A View of Source Text for Diversely Configurable Software", Technical Report CS-79-28, Dept. of Computer Science, University of Waterloo, 1979.
- [7] K. Kwang and F.A. Briggs, "Computer Architecture and Parallel Processing", McGraw-Hill Book Company, 1984.
- [8] Multibus I, IEEE 796.
- [9] "Standard Specifications For Versatile Backplane Bus", Draft 1.0, IEEE Microprocessor Standards Committee, P1014 Standard Committee, Feb. 1985.
- [10] W.M. Gentleman, "Message Passing Between Sequential Processes: The Reply Primitive and the Administrator Concept", Software Practice and Experience, Vol. 11, pp. 433-466 (1981).
- [11] B.W. Boehm, "Software Engineering Economics", Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632, 1981.