PROCEEDINGS

Southern African Computer Symposium 1991

6th de Rekenaarsimposium van Suider Afrika 1991

DE OVERBERGER HOTEL, CALEDON

2 - 3 JULY 1991

SPONSORED BY

ISM

FRD

GENMIN

EDITED BY

M H Linck

DEPARTMENT OF COMPUTER SCIENCE • UNIVERSITY OF CAPE TOWN

PROCEEDINGS / KONGRESOPSOMMINGS

6th
SOUTHERN AFRICAN COMPUTER SYMPOSIUM

6de
SUIDELIKE-AFRIKAANSE REKENAARSIMPOSIUM

De Overberger Hotel, Caledon

2 - 3 JULY 1991

SPONSORED by

ISM
FRD
GENMIN

EDITED by

M H LINCK

Department of Computer Science

University of Cape Town

TABLE OF CONTENTS

FOREWORD

The 6th Computer Symposium, organised under the auspices of SAICS, carries on the tradition of providing an opportunity for the South African scientific computing community to present research material to their peers.

It was heartening that 31 papers were offered for consideration. As before all these papers were refereed. Thereafter a selection committee chose 21 for presentation at the Symposium.

Several new dimensions are present in the 1991 symposium:

* The Symposium has been arranged for the day immediately after the SACLA conference.

* It is being run over only 1 day in contrast to the 2-3 days of previous symposia.

* I believe that it is first time that a Symposium has been held outside of the Transvaal.

* Over 85 people will be attending. Nearly all will have attended both events.

* A Sponsorship package for both SACLA and the Research Symposium was obtained. (This led to reduced hotel costs compared to previous symposia)

A major expense is the production of the Proceedings of the Symposium. To ensure financial soundness authors have had to pay the page charge of R20 per page.

A thought for the future would be consideration of a poster session at the Symposium. This could provide an alternative approach to presenting ideas or work.

I would sincerely hope that the twinning of SACLA and the Research Symposium is considered successful enough for this combination survive. As to whether a Research Symposium should be run each year after SACLA, or only every second year, is a matter of need and taste.

A challenge for the future is to encourage an even greater number of MSc & PhD students to attend the Symposium. Unlike this year, I would recommend that they be accommodated at the same cost as everyone else. Only if it is financially necessary should the sponsored number of students be limited.

I would like to thank the other members of the organising committee and my colleagues at UCT for all the help that they have given me. A special word of thanks goes to Prof. Pieter Kritzinger who has provided me with invaluable help and ideas throughout the organisation of this 6th Research Symposium.

M H Linck
Symposium Chairman

SYMPOSIUM CHAIRMAN

M H Linck, University of Cape Town

ORGANISING COMMITTEE

D Kourie, Pretoria University.

P S Kritzinger, University of Cape Town.

M H Linck, University of Capè Town.

SPONSORS

ISM

GENMIN

FRD

LIST OF REFEREES FOR 6th RESEARCH SYMPOSIUM

NAME	INSTITUTION
Barnard, E	Pretoria
Becker, Ronnie	UCT
Berman S	UCT
Bishop, Judy	Wits
Berman, Sonia	UCT
Brink, Chris	UCT
Bodde, Ryn	Networks Systems
Bornman, Chris	UNISA
Bruwer, Piet	UOFS
Cherenack, Paul	UCT
Cook Donald	UCT
de Jaeger, Gerhard	UCT
de Villiers, Pieter	Stellenbosch
Ehlers, Elize	RAU
Eloff, Jan	RAU
Finnie, Gavin	Natal
Gaynor, N	AECI
Hutchinson, Andrew	UCT
Jourdan, D	Pretoria
Kourie Derrick	Pretoria
Kritzinger, Pieter	UCT
Krzesinski, Tony	Stellenbosch
Laing, Doug	ISM
Labuschagne, Willem	UNISA
Levy, Dave	Natal

MacGregor, Ken	UCT
Machanick, Philip	Wits
Mattison Keith	UCT
Messerschmidt, Hans	UOFS
Mutch, Laurie	Shell
Neishlos, N	Wits
Oosthuizen, Deon	Pretoria
Peters Joseph	Simon Fraser
Ram, V	Natal, Pmb.
Postma, Stef	Natal, Pmb
Rennhackkamp, Martin	Stellenbosch
Shochot, John	Wits
Silverberg, Roger	Council for Mineral Technology
Smit, Riël	UCT
Smith, Dereck	UCT
Terry, Pat	Rhodes
van den Heever, Roelf	UP
van Zijl, Lynette	Stellenbosch
Venter, Herman	Fort Hare
Victor, Herna	Stellenbosch
von Solms, Basie	RAU
Wagenaar, M	UPE
Wentworth, Peter	Rhodes
Wheeler, Graham	UCT
Wood, Peter	UCT

6TH RESEARCH SYMPOSIUM - 1991

FINAL PROGRAM

TUESDAY 2nd July 1991

10h00 - 13h00 **Registration**

13h00 - 13h50 **PUB LUNCH**

14h00 - 15h30 SESSION 1A

Venue: Hassner

Chairman: Prof Basie von Solms

14h00 - 14h30
"A value can belong to many types. "
B H Venter, University of Fort Hare

14h30 - 15h00
"A Transputer Based Embedded Controller Development System"
M R Webster, R G Harley, D C Levy & D R Woodward, University of Natal

15h00 - 15h30
"Improving a Control and Sequencing Language"
G Smit and C Fair, University of Cape Town

SESSION 1B

Venue: Hassner C

Chairman:Prof Roelf v d Heever

14h00 - 14h30
"Design of an Object Orientated Framework for Optimistic Parallel Simulation on Shared-Memory Computers" P Machanick, University of Witwatersrand

14h30 - 15h00
"Using Statecharts to Design and Specify the GMA Direct-Manipulation User Interface" L van Zijl & D Mitton, University of Stellenbosch

15h00 - 15h30
"Product Form Solutions for Multiserver Centres with Heirarchical Classes of Customers" A Krzesinski, University of Stellenbosch and R Schassberger, Technische Universität Braunschweig

15h30 - 16h00 **TEA**

16h00 - 17h30 SESSION 2A

Venue: Hassner

Chairman: Prof Derrick Kourie

16h00 - 16h30
"A Reusable Kernel for the Development of Control Software" W Fouché and P de Villiers, University of Stellenbosch

16h30 - 17h00
"An Implementation of Linda Tuple Space under the Helios Operating System" P G Clayton, E P Wentworth, G C Wells and F de-Heer-Menlah, Rhodes University

17h00 - 17h30
"The Design and Analysis of Distributed Virtual Memory Consistency Protocols in an Object Orientated Operating System" K MacGregor, University of Cape Town & R Campbell, University of Illinois at Urbana-Champaign

19h30 **PRE-DINNER DRINKS**

20h00 **GALA CAPE DINNER**
(Men: Jackets & ties)

WEDNESDAY 3rd July 1991

7h00 - 8h15 BREAKFAST

8h15 - 9h45 SESSION 3A

Venue: Hassner

Chairman: Assoc Prof P Wood

8h15 - 8h45
"Concurrency Control Mechanisms for Multidatabase Systems" A Deacon, University of Stellenbosch

8h45 - 9h15
"Extending Local Recovery Techniques for Distributed Databases" H L Victor & M H Rennhackkamp, University of Stellenbosch

9h15 - 9h45
"Analysing Routing Strategies in Sporadic Networks" S Melville, University of Natal

SESSION 3B

Venue: Hassner C

Chairman: Prof G Finnie

8h15 - 8h45
The Design of a Speech Synthesis System for Afrikaans" M J Wagener, University of Port Elizabeth

8h45 - 9h15
"Expert Systems for Management Control: A Multiexpert Architecture" V Ram, University of Natal

9h15 - 9h45
"Integrating Simularity-Based and Explanation-Based Learning" G D Oosthuizen and C Avenant, University of Pretoria

9h45 - 10h15 TEA

10h15 - 11h00 SESSION 4

Venue: Hassner

Chairman: Prof P S Kritzinger
Invited paper: E Coffman

11h00 - 11h10 BREAK

11h10 - 12h40 SESSION 5A

Venue: Hassner

Chairman: Prof C Bornman

11h10 - 11h40
"Efficient Evaluation of Regular Path Programs"
P Wood, University of Cape Town

11h40 - 12h10
"Object Orientation in Relational Databases"
M Rennhackkamp, University of Stellenbosch

12h10 - 12h40
"Building a secure database using self-protecting objects" M Olivier and S H von Solms, Rand Afrikaans University

SESSION 5B

Venue: Hassner C

Chairman: Prof A Krzesinski

11h10 - 11h40
"Modelling the Algebra of Weakest Preconditions"
C Brink & I Rewitsky, University of Cape Town

11h40 - 12h10
"A Model Checker for Transition Systems"
P de Villiers, University of Stellenbosch

12h10 - 12h40
"A New Algorithm for Finding an Upper Bound of the Genus of a Graph"
D I Carson and O R Oellermann, University of Natal

12h45-12h55 GENERAL MEETING of RESEARCH SYMPOSIUM ATTENDEES

Venue: Hassner

Chairman: Dr M H Linck

13h00 - 14h00 LUNCH

FINIS 6th COMPUTER SYMPOSIUM

PAPERS

of the

6TH RESEARCH SYMPOSIUM

A Reusable Kernel For the Development of Control Software

W. Fouché and P.J.A. de Villiers
Institute for Applied Computer Science
University of Stellenbosch, Stellenbosch 7600

Abstract

A kernel has been developed which can be used to simplify the development of control software. It can be used to control several physical machines which are interconnected via a fast local area network. One or more *virtual* machines are simulated on each physical machine. The kernel was used to implement an operating system consisting of a file server, a name server and a shell. However, it can also be used to support control software for embedded systems. The performance of the kernel has been measured and found to be comparable to other kernels.

1 Introduction

The long term goal of the HYBRID project is to develop control software for distributed hardware. One method which simplifies this task is to develop *reusable software modules*. In this respect a programming language which directly supports modules is invaluable. We chose to use Modula-2[20]. The programming of bare hardware is a time consuming but necessary part of the development of most control systems. Benefits can be obtained by packaging such low-level code as a reusable *kernel* so that a designer can use it without necessarily understanding its implementation details. The design of electronic systems was simplified significantly when engineers packaged reusable designs as integrated circuits. Any integrated circuit can be used as a "black box" as long as its function is understood properly, its internal details being irrelevant. We believe that an efficient and reliable kernel can do the same for the design of control software. The design and implementation of a reusable, reliable and efficient kernel was therefore the major initial activity of this project.

2 A Kernel Supporting Virtual Machines

The HYBRID kernel provides the basic functionality needed to implement control systems: support for multiple processes, memory management, interprocess communication and operations on peripheral devices. To simplify a control system it is necessary to decompose it into simpler subsystems. An efficient communication facility is therefore needed to enable these subsystems to cooperate. Communication must be based on simple but general principles in order to avoid subtle

errors. CSP[11] presents a theory which can be used as a basis for the orderly design of concurrent systems. Concurrent processes are only allowed to communicate by exchanging messages according to simple well-defined rules. The success of the transputer provides evidence that CSP can be used as a basis for the design of practical systems. Although transputer networks can host control software efficiently, it is restrictive that the number of transputers in a given network is fixed.

The HYBRID kernel transforms different physical machines, interconnected by a high performance local area network, into several communicating virtual machines (VMs) which are similar in many ways to transputers. VMs execute processes which communicate according to the principles of CSP. Since VMs can be created dynamically the HYBRID kernel relaxes a constraint of transputer networks—a fixed number of transputers.

The kernel makes simple and elegant designs feasible because a *virtual* machine can be dedicated to each autonomous task. If a task does not need the power of a dedicated physical machine, more than one VM can be executed by the same physical machine. It is a matter of separating concerns: we believe that using the concept of VMs, each dedicated to a specific task, can lead to elegant designs, while multiplexing a *physical* machine among several VMs is a separate matter which is handled by the kernel. *A designer can thus ignore the underlying hardware and think in terms of communicating VMs.*

3 Design of the Kernel

A detailed discussion of the internal structure of the HYBRID kernel in which the various design decisions are motivated is presented in [9]. Here we present only an overview of the most important concepts. The design of the kernel is conservative rather than innovative, exploiting successful concepts found in existing systems such as Amoeba[14, 19, 15], Chorus[10, 18, 1], Mach[17, 16] and V[6, 4, 5]. These systems implement similar services by using different techniques as dictated by the experience and beliefs of their designers.

In order to enable a system designer to think in terms of abstract concepts alone, the kernel hides the properties of the underlying physical machine. Another important feature of the kernel is scalability—if better performance is needed another processor can be added without changing the logical design of the control system which is designed in terms of *virtual* machines. A VM is an abstract machine with the following properties:

- It has a separate protected address space which contains code and data.
- A VM can execute one or more (light-weight) processes which share the same address space.
- Processes communicate by sending and receiving messages.

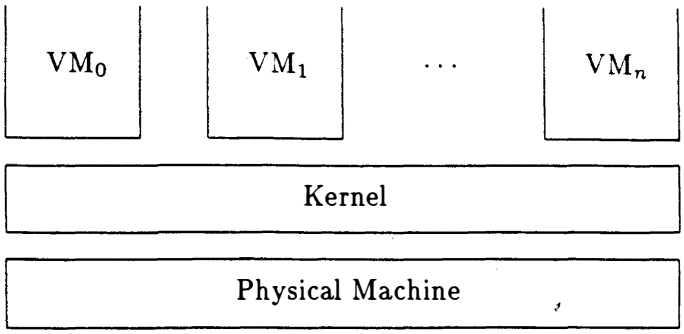

Figure 1: Multiple VMs Supported by a Single Physical Machine

Figure 1 shows a single physical machine supporting several VMs. To maintain system integrity the kernel prevents one VM from accessing the address space of another. This makes it impossible for any VM to accidentally (or maliciously) modify the private memory area of another. Hardware support for memory protection is thus essential.

A process is represented in the kernel by its *process descriptor* which is used to store its current state. Process descriptors of processes which are ready to run are linked to form a *process queue*. In a similar way a VM is represented by its VM descriptor. Each VM descriptor contains a pointer to its associated process queue. VM descriptors of VMs which can be enabled are linked to form a *VM queue*. The scheduler first selects a VM to activate and then selects an appropriate process to run. VMs are time-sliced and may have different priorities, but a process is allowed to run until an interprocess communication (IPC) operation is invoked or until the time-slice of its VM expires.

Unbuffered synchronous message passing is used to exchange information between processes via ports. Ports are global identifiers used to route messages to their destinations. We assume most messages to be short. Bershad *et al.*[3] found that nearly 80% of all messages transmitted in Topaz, the operating system of the Firefly multiprocessor workstation developed at DEC SRC, are more or less 32 bytes in length. Thus it was decided to use *copying* to transfer messages between processes on the same machine. Alternatively, page remapping techniques[8] can be used, but the extra complexity seems to be worthwhile only for large messages. Three IPC operations are defined: **Transaction**, **ReceiveRequest** and **SendReply**. **Transaction** is used to send a request message to a process and to receive the associated reply message. **ReceiveRequest** receives a message and **SendReply** returns a reply message to a process. A sender process, making use of the **Transaction**

primitive to send a message, remains suspended until the request message has been accepted and a reply message returned. Similar models for interprocess communication have been found to work well in the kernels of the Amoeba[15] and V[5] systems.

All device servers (peripheral drivers) are coded in a similar way and it is necessary to understand just a few basic principles in order to add a server for a new peripheral device. A device server receives requests which are entered in a request queue. While requests are available, one is selected from this queue and the appropriate peripheral operation is started. The device server is then deactivated by calling the scheduler to resume the next ready process. The interrupt which signals completion of the peripheral operation reactivates the device server in order to start the next peripheral operation.

It can thus be seen that interrupt handling, process scheduling and message passing are closely interconnected although this is of no concern to a programmer at the user level—a VM is a clearly defined deterministic machine which supports a natural execution environment for one or more cooperating sequential processes. The IPC facility provides a simple means of process synchronisation.

Although the development of the HYBRID kernel was motivated primarily by a requirement to implement an operating system for distributed hardware, we later realised that with minor modifications the kernel could also support other classes of control software.

4 A VM-based Operating System

The operating system which was developed as a first application of the kernel is used for research and educational purposes. Since an operating system is an example of a complex control system this application represents a reasonable test of the kernel. The system is based on the client-server concept and was deliberately kept as simple as possible, its various services being made accessible via the standard communication facilities offered by the kernel. Efficient servers can be designed as a group of cooperating processes which are supported naturally by a VM. Clients request servers to perform operations on their behalf by transmitting messages. Servers send reply messages to inform clients of the outcome of requested operations or to return requested data to them. Sophisticated services may involve more than one server VM implying that the functionality of the operating system is distributed across several VMs.

4.1 Implementing a File Service

One of the more important services provided by an operating system is a *file service* which is implemented by a file system. The internal details of the file system can be hidden from the user by separating the logical structure of the file system from its physical structure. This is done by partitioning the file access facilities into two

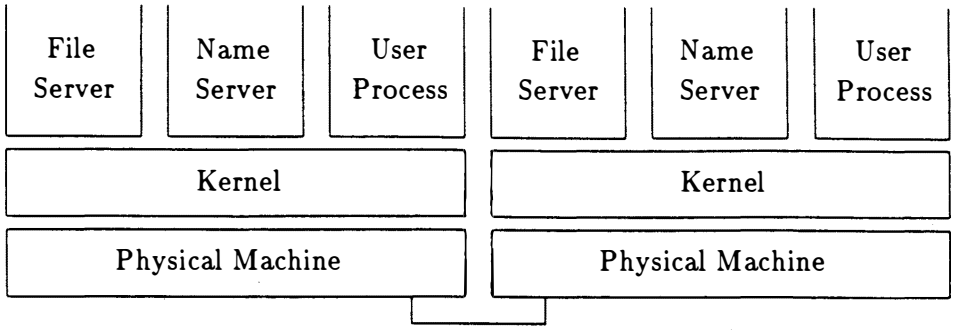

Figure 2: A Distributed System Based on VMs

functions, namely, the actual organisation of the file in terms of records on the disk as maintained by the *file server* and the file naming and protection mechanisms which are provided by the *name server*. The file server and name server are each implemented by a VM as illustrated in Figure 2.

The file server VM maintains a set of files, each file consisting of a linked list of disk blocks. A unique identifier (UID) is associated with every file. The file server implements operations on files such as *read*, *open* and *write*.

Clients refer to files by name. A name is a text string such as "/usr/joe/report5" which makes sense to a user. Such a file name is translated by the name server to a UID which is more convenient for the file system to handle. Directory structures are thus implemented by the name server. For reasons of efficiency there can be more than one name server in a distributed environment as long as a consistent database of names is maintained. The same resource can also be known to different users by different names if that is convenient.

4.2 User-level Processes

A user starts a session by logging in. This creates a single VM (with an associated screen and keyboard) which executes a single process—a Unix-like shell—capable of interpreting commands entered from the keyboard. Another session can be started by entering a command to create an additional VM. This may be repeated a predetermined number of times. For instance a user can have a VM (session) for editing and another for compiling and testing a program. The keyboard and screen can be transferred from one VM to another to allow a user to switch between sessions.

A schematic view of the distributed system consisting of the kernel resident on each physical machine and an operating system supported by various VMs is given in Figure 2.

5 Performance of the Kernel

In order to determine the efficiency of the kernel a series of test programs were constructed. All tests were conducted on machines with a 25 MHz Intel-386 processor. Remote communication depends to a large extent on hardware speeds, but initial tests seem to indicate that a basic speed of 3 Mbits per second over an ethernet of 10 Mbits per second is attainable. Here we concentrated on measuring the efficiency of the kernel software: local interprocess communication overhead and the cost of the process and VM management facilities.

5.1 Interprocess Communication Facility

Two sets of test programs were used to evaluate the efficiency of the IPC facility. Firstly, messages of different lengths were transmitted from a process on one VM to a process on *a different VM*, both VMs residing on the same physical machine. The receiving process acknowledges receipt of the message by sending back an empty reply message. Such an interaction between two processes is called a *message transaction*[13] which is a simple form of a remote procedure call. Secondly, messages were transmitted between processes supported by *the same VM*.

An initial (perhaps naive) implementation of IPC yielded the performance figures shown in Table 1. The delay to transmit a single message is given in milliseconds and the communication bandwidth is given in terms of both Kbytes per second and a percentage of the maximum speed at which the machine can copy data from one memory area to another.

The reason why message passing between different VMs was slow in the initial implementation of the kernel is that messages were copied first from the sending VM to the kernel and then from the kernel to the receiving VM. Many context switches were necessary since long messages were copied one 4Kbyte physical memory page at a time. It was much simpler to transfer messages between processes on the same VM. In this case no context switches were necessary and the whole message could be copied in one go. Since the kernel is meant to support control software designed as several communicating VMs, it was mandatory to improve the efficiency of message passing between processes located on different VMs.

The hardware of the 386 allows a VM to have a 4Gbyte address space, each VM having its own page table. By reducing the maximum allowable size of the address space of a VM to 2Gbytes, only the lower half of the page table of each VM is used. The upper half of each page table can now be used to gain access to the source or destination address space of a message, whichever is appropriate. This technique allows us to use exactly the same code to handle message passing between processes on the same VM and processes on different VMs. It significantly improved the efficiency of message passing between processes on different VMs as shown in Table 2. Although *the same* code is used to transfer messages between processes on the same VM and processes on different VMs, the performance is different. If it is

IPC speed for processes on different VMs			
Message Size (Kbytes)	Delay (milliseconds)	Bandwidth (Kbytes per second)	Bandwidth expressed as a % of max. copy speed
0	2.69	0	0.0
1	3.36	298	2.0
2	4.00	499	3.3
4	5.27	759	5.0
8	7.93	1009	6.6
16	13.17	1215	8.0
32	23.65	1353	8.9

IPC speed for processes on the same VM			
Message Size (Kbytes)	Delay (milliseconds)	Bandwidth (Kbytes per second)	Bandwidth expressed as a % of max. copy speed
0	0.51	0	0.0
1	0.77	1299	8.5
2	1.03	1942	12.7
4	1.55	2572	16.8
8	2.60	3083	20.2
16	4.68	3419	22.4
32	8.84	3618	23.7

Table 1: Initial Implementation of IPC

kept in mind that page table entries are cached by the 386, this can be explained. The cache (*translation lookaside buffer* or TLB) is a buffer of fixed size and it is flushed on each context switch. When a message is transmitted between processes on the same VM a context switch is unnecessary and the TLB helps to speed up the task of copying the message. However, when messages are transmitted between processes on different VMs it is awkward to avoid a context switch and the extra amount of testing required to do so is not considered worthwhile. However, this possibility has not yet been explored thoroughly.

By carefully rewriting the procedure which is used to copy data in assembly language, an additional improvement in the transfer rate was possible—see Table 3. With messages of 32K in length a data transfer rate of 12400 KBytes per second can be sustained between two processes on the same VM. This is within 80% of the maximum speed at which the test machine can transfer data from one location to another in memory.

5.2 VM and Process Management Facilities

The creation of a new VM requires the establishment of a virtual address space, the creation of a process to execute code and the initialization of the VM. The minimum time required to create and terminate a VM is 12.1 milliseconds. This includes the time needed to: install an executable program on a VM, create and terminate a

IPC speed for processes on different VMs			
Message Size (Kbytes)	Delay (milliseconds)	Bandwidth (Kbytes per second)	Bandwidth expressed as a % of max. copy speed
0	0.80	0	0.0
1	1.24	810	5.3
2	1.66	1201	7.9
4	2.49	1606	10.5
8	4.16	1921	12.6
16	7.53	2125	13.9
32	14.24	2246	14.7

Table 2: Copying data directly

IPC speed for processes on different VMs			
Message Size (Kbytes)	Delay (milliseconds)	Bandwidth (Kbytes per second)	Bandwidth expressed as a % of max. copy speed
0	0.78	0	0.0
1	0.89	1117	7.3
2	1.00	2000	13.1
4	1.22	3292	21.6
8	1.63	4893	32.1
16	2.47	6465	42.4
32	4.16	7692	50.4

IPC speed for processes on the same VM			
Message Size (Kbytes)	Delay (milliseconds)	Bandwidth (Kbytes per second)	Bandwidth expressed as a % of max. copy speed
0	0.47	0	0.0
1	0.55	1818	11.9
2	0.62	3252	21.3
4	0.75	5333	34.9
8	1.01	7921	51.9
16	1.53	10458	68.5
32	2.58	12403	81.3

Table 3: Hand-optimised implementation

System	ratio of RPC to PC
V	182:1
Amoeba	200:1
HYBRID	295:1
Mach	416:1

Table 4: Efficiency of RPC compared to a simple procedure call

process and to allocate and deallocate a small address space. In general this figure depends on the size of the VM because the actual time required to create a new VM depends on the size of its address space and the size of the executable program.

The total time taken to create and terminate a process amounts to 0.55 milliseconds. This includes the allocation of a new process descriptor and stack for the process in the kernel, a run-time stack in user space, two context switches and deallocation of memory assigned to the process.

5.3 Comparing Simple and Remote Procedure Calls

The kernel has been in operation for some time and although some improvements are still possible, its performance is considered to be acceptable. Although we tried to make IPC operations as efficient as possible, remote procedure calls remain expensive when compared to simple procedure calls. The main reason for this seems to be the interaction with the scheduler. An interesting idea to be explored in this respect is *upcalls*[7]. Comparing the performance of different systems is normally meaningless unless the same hardware is used. Since this is seldom the case in practice, the best we could do was to compare the *speed ratio* between remote procedure calls and simple procedure calls. Similar measurements are available for a number of well-known kernels. As shown in Table 4 the HYBRID kernel is comparable in efficiency to the Amoeba, Mach and V kernels in this respect.

6 Conclusion

Several distributed systems were developed since the mid 1970's when RIG[2, 12], the first major distributed system, became operational. All these systems were based on small message based kernels. Chorus, Mach, Amoeba and the Stanford V system are representative of the current state of the art. Although they reflect different philosophies regarding communication and process management, all proved to be successful. The first goal of the HYBRID project was to exploit the best ideas from these experimental systems to produce a kernel which is useful as a practical tool to simplify the implementation of control software. Because most designers of control systems find it natural to work with *machines* of some kind, we

have designed a kernel which transforms the physical machine which is awkward to program into one or more virtual machines which are easier to program. VMs which can support multiple processes, are similar to *team spaces* in the V system, *tasks* in Mach and *process clusters* in Amoeba. VMs can be used to group cooperating processes together to optimise interprocess communication. The problem is to find the right abstractions which can hide irrelevant detail of the underlying physical machine *without hiding its power*. Whether VMs can do this will have to be seen, but design attempts undertaken so far are encouraging.

The functionality defined by the VM abstraction decouples a control system from the underlying hardware. It is therefore possible to use the same control software on different hardware platforms by porting the kernel.

Preliminary experiments led to a simple kernel for a Data General minicomputer by May 1989. We gained valuable experience regarding process and peripheral management during these early experiments. Relying on this experience we redesigned the kernel and a functionally complete HYBRID kernel for an Intel 386-based machine has been operational since the end of 1990. To date, the kernel has been used to develop a distributed operating system for research and educational purposes. Its performance is comparable to a commercial Unix system executing on the same hardware. The kernel has now been running without problems for about six months and appears to be reliable. Although currently used to support an operating system the kernel can be reused to support other applications. For example, a terminal concentrater can be designed as a number of cooperating VMs supported by a single physical machine.

7 Acknowledgements

The first device drivers for the kernel were written by Harry Lewis. As the first user of the kernel he detected a few subtle coding errors. The operating system which represented the first test of the kernel was developed jointly by William Howard and Harry Lewis.

References

[1] V. Abrossimov and M. Rozier, "Generic Virtual Memory Management for Operating System Kernels", in *Proceedings of the 12th ACM Symposium on Operating System Principles*, pp. 123–136, Published as ACM Operating Systems Review, Vol. 23, No. 5, December 1989.

[2] J. E. Ball, J. A. Feldman, J. R. Low, R. F. Rashid, and P. D. Rovner, "RIG, Rochester's Intelligent Gateway: System Overview", *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 321–328, December 1976.

[3] B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy, "Lightweight Remote Procedure Call", Technical Report 89-04-02, Department of Computer Science, University of Washington, April 1989.

[4] D. R. Cheriton, "An Experiment using Registers for Fast Message-Based Interprocess Communication", *ACM Operating Systems Review*, vol. 18, no. 4, pp. 12–20, October 1984.

[5] D. R. Cheriton, "The V Distributed System", *Communications of the ACM*, vol. 31, no. 3, pp. 314–333, March 1988.

[6] D. R. Cheriton and W. Zwaenepoel, "The Distributed V Kernel and its Performance for Diskless Workstations", *Proceedings of the 9th ACM Symposium on Operating System Principles*, pp. 129–140, 1983.

[7] D. C. Clark, "The structuring of systems using upcalls", *ACM Operating Systems Review*, vol. 19, no. 5, pp. 171–180, December 1985.

[8] R. Fitzgerald and R. F. Rashid, "The Integration of Virtual Memory Management and Interprocess Communication in Accent", *ACM Transactions on Computer Systems*, vol. 4, no. 2, pp. 147–177, May 1986.

[9] W. Fouché, "The HYBRID kernel", Tech. Rep. ITR-90-01-00, Institute for Applied Computer Science, University of Stellenbosch, April 1990.

[10] M. Guillemont, "The Chorus Distributed Operating System: Design and Implementation", in *Local Computer Networks*, (P. C. Ravasio, G. Hopkins, and N. Naffah, eds.), pp. 207–223, Proceedings of the IFIP TC 6 International In-Depth Symposium on Local Computer Networks, Florence, Italy, (19-21 April), North-Holland Publishing Company, 1982.

[11] C. A. R. Hoare, *Communicating Sequential Processes. International Series in Computer Science*, Englewood Cliffs, New Jersey: Prentice-Hall International, 1985.

[12] K. A. Lantz, J. A. Feldman, and R. F. Rashid, "Rochester's Intelligent Gateway", *Computer*, vol. 15, no. 10, pp. 54–68, October 1982.

[13] S. J. Mullender, "Distributed Operating Systems", *Computer Standards and Interfaces*, vol. 6, no. 1, pp. 37–44, 1987.

[14] S. J. Mullender and A. S. Tanenbaum, "The Design of a Capability-Based Distributed Operating System", *The Computer Journal*, vol. 29, no. 4, pp. 289–299, 1986.

[15] S. J. Mullender, G. van Rossum, A. S. Tanenbaum, R. van Renesse, and H. van Staveren, "Amoeba: A Distributed Operating System for the 1990s", *IEEE Computer*, vol. 23, no. 5, pp. 44–53, May 1990.

[16] R. Rashid, A. Tevanian, M. Young, D. Golub, R. Baron, D. Black, W. J. Bolosky, and J. Chew, "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures", *IEEE Transactions on Computers*, vol. 37, no. 8, pp. 896–907, August 1988.

[17] R. F. Rashid, "From RIG to Accent to Mach: The Evolution of a Network Operating System", *Proceedings of the ACM/IEEE Computer Society Fall Joint Conference*, pp. 1128–1137, November 1986.

[18] M. Rozier and J. L. Martins, "The CHORUS Distributed Operating System: Some Design Issues", in *Distributed Operating Systems: Theory and Practice*, (Y. Paker, J. Banatre, and M. Bozyigit, eds.), pp. 261–287, Springer-Verlag, 1987.

[19] R. Van Renesse, H. Van Staveren, and A. S. Tanenbaum, "The Performance of the Amoeba Distributed Operating System", *Software—Practice and Experience*, vol. 19, no. 3, pp. 223–234, March 1989.

[20] N. Wirth, *Programming in Modula-2*. Springer-Verlag, 2 ed., 1983.