AUUGN

Australian Unix systems User Group Newsletter

Volume 7

Number 1

Registered by Australia Post Publication No. NBG6524

The Australian UNIX* systems User Group Newsletter
Volume 7 Number 1

October 1986

CONTENTS

* UNIX is a trademark of AT&T Bell Laboratories

Editorial

Welcome to my first issue as editor of the Australian UNIX systems Users Group Newsletter or the AUUGN (pronounced *organ*). This issue contains papers that were presented at the AUUG Winter '86 meeting that was held at the ANU, Canberra. The issue provides all Members, reguardless of whether they attended, a record of the proceedings.

My task of compiling this issue of the Newsletter has been made easier by the Meeting generating most of the content. This will not always be the case. I have been asked by AUUG Management Committee to produce a Newsletter regularly, every two months, and the next Newsletter in December will look very thin unless you help. The Newsletter needs material that is of interest of AUUG members, so here are a few areas that you could help by sending:

- news that is of general interest.
- a review of a book.
- a paper on about a project in which you involved.
- an article on an area in which you have expertise.

Please let me know, if you have have a contribution to make to the AUUGN,

Special thanks to the retiring editor, Peter Ivanov for producing a Newsletter I always found interesting and worth reading.

I look forward to hearing from you,

John Carey

Memberships and Subscriptions

The new Membership forms, referred to in the Management Committee Report, can be found at the end of this issue. AUUG Membership includes subscription to this Newsletter. The AUUGN is now not available by subscrition, without Membership.

All correspondence concerning membership of the AUUG should be addressed to:

The AUUG Membership Secretary,
P.O. Box 366,
Kensington, N.S.W. 2033.
Australia.

Next AUUG Meeting

The next meeting will be held at Telecom Research in Clayton, Victoria. Futher details will be provided in the next issue.

Contributions

The Newsletter is always of need good material, so please send me your contributions. My address is printed on the inside back cover.

Disclaimer

Opinions expressed by authors and reviewers are not necessarily those of the Australian UNIX systems User Group, its Newsletter or its editorial committee.

President's Message

Firstly, I would like to say "thank you" to several people for their contributions to AUUG.

- As retiring and foundation President of AUUG, **John Lions** has been instrumental in placing the group on a sound constitutional base, and charting the delicate course for navigating the group from the small friendly group of zealots that began meeting around 1977, into a large group of friendly Unix devotees and users that now constitute our membership. During this decade of change, many of the same faces have remained, albeit now less hairy (kre not withstanding), more frequently washed and better dressed (there are notable exceptions here, I agree); I am sure that this stability of personnel and John's leadership have been instrumental in forming a group that operates in a co-operative and mutually supportive manner.
- Retiring management committee members, **Piers Lauder** and **Greg Rose** are also foundation members who have devoted considerable energies to AUUG and its members over the past ten years.
- **Peter Ivanov**, has relinquished the editorship of the Newsletter, and this issue is the first under **John Carey**'s control. Like all AUUG office-bearers, the Newsletter editor is a volunteer, and very few people would appear willing to take on the task of beating Unix users over the head with e-mail until they produce promised copy. We have been lucky that Peter has been willing to devote so much time to the Newsletter, and trust that John will receive more support to reduce the burden of this essential task. To both of you, "thanks".
- The participation of **Lionel Singer** and **Chris Campbell** on the management committee represents not only "new blood", but also the is a reflection of changes in the composition of AUUG membership. So far AUUG has managed to address the needs of a broad spectrum of Unix users (commercial end-user, systems builder, educator and researcher), and the election of non-academics onto the management committee provides further insurance against the danger of splitting our membership (as in the USENIX *vs* /usr/group wars).
- And finally, a special "thank you" to all the folks in Canberra for the organization of the last meeting.

With this issue of the Newsletter, we are commencing a policy of tight adherence to copy and publication deadlines; the Newsletter will appear at approximately regular intervals, six times per year. If the issues end up being thin, that is a consequence of a failure on YOUR part (not on the part of the editor) – so keep those cards and letters rolling in folks. I would imagine that

cerebellum: Segmentation fault – core dumped.

is about the only legitimate excuse for someone at *your* site not contributing to the next Newsletter.

I would appreciate receiving comments either as direct e-mail, or as a news discussion in aus.auug, or via newsletter contributions that suggest potential activities for AUUG sponsorship. Here is a starter list to warm up the flame-throwers,

- Negotiating to make ACSnet more widely and cheaply available.
- Bulk subscriptions to /usr/group or USENIX publications, e.g. the /usr/group software catalog.
- Poking our noses into standardization efforts – ANSI C, Xopen, X.400 and MHS, IEEE POSIX, etc.
- Running workshops as commercial ventures with profits shared between presenters and AUUG.

Ken J. McDonell
kenj@moncsbruce.oz

Minutes of the AUUG Management Committee Meeting
September 1, 1986

1. The meeting opened at 17:15. Present were Chris Campbell (CC), Robert Elz (KRE), John Lions (JL), Chris Maltby (CM), Tim Roper (TR), and Lionel Singer (LS). An apology was received from Ken McDonell (KENJ). Also present were John Carey (JC), Glenn Huxtable (GH), Steve Jenkin (SJ), Greg Rose (GR), Greg Webb (GW), Rod Wier (RW), and Peter Wishart (PW).
2. In the absence of the president, JL was elected chairman of the meeting.
3. Moved (JL, seconded KRE) **That the required notice of the meeting be waived.** Carried (6-0).
4. The minutes of the previous meeting (February 1986) were read.
5. Moved (JL, seconded CM). **That the minutes be referred to the secretary for further work, and approval be deferred until the next meeting.** Carried (6-0)

   JL noted that all of the officers of the previous committee had been overworked, and no blame for failure to complete assigned duties should be assigned to anyone in particular. It was also noted that some of the committee members have additional notes from the previous meeting that could be used to assist in completing the minutes.
6. Business arising out of the minutes
   - KRE noted that the document giving guidance on how to hold a meeting had been updated and distributed. PW agreed to update it further based on his experiences with the current meeting.
   - GR indicated that he had checked on the availability of bankcard, etc, and that it would be easy to obtain. Estimated cost approximately $100 plus 5% on each transaction.
   - Moved (TR, seconded LS) **That the treasurer should proceed with obtaining the necessary authorities and equipment, and be authorised to expend necessary funds.** Carried (6-0).

     LS noted that bank clerks don't always know what they are doing, must be sure to do this properly.
7. No presidents report was given, as the president was absent.
8. The secretary's report was tabled.

   There was some discussion on the correspondence with Ambrose. LS suggested that no action should have been taken without a fee being collected. GH asked whether this type of letter should appear in AUUGN. A policy on advertising was noted as something that was desirable to have.

GR (past secretary) indicated that there had been nothing but routine matters in the past year. A newsletter exchange has been established with UNIGRAM/X. No progress has been made on reciprocal arrangements.

Moved (CM, seconded CC) **That the secretaries report be accepted.** Carried (6-0).

9. The balance sheet was tabled.

   The treasurer (CM) indicated:

   - That more funds could be moved to the high interest account.
   - That something would have to be done to obtain funds owed by the UnixWorld conference.
   - That the costs of AUUG were rising, postage rates have increased, and the relevant sales tax had increased from 5% to 20%.
   - The weight of each issue or the newsletter should be carefully watched as this affects the postage cost.

   LS queried some of the accounting, and indicated that he could have the balance sheet redone.

   GH apologised for the state of the balance sheet of the Perth AUUG meeting.

   It was suggested that meeting finances be kept entirely separate from general finances.

   LS asked who would contact Stephen Moore of Computerworld about the UnixWorld debt. JL volunteered, as did LS and CC. JL to act initially.

   Moved (KRE, seconded TR) **That the treasurer's report be accepted with alterations as noted.** Carried (6-0).

10. The meeting organiser's report was given by PW. He indicated that the advance registration was 110 (including 8 who registered on site the previous day) and while he did not yet have figures on the number of attendees who had registered that day, he anticipated total attendance to be about 140.

    It was noted that Mike Banahan's costs had not yet been reimbursed. PW indicated that the meeting could cover them. JL and PW to investigate.

    TR suggested that Banahan be given choice of currency.

    PW indicated that this was the first meeting to have had a programme committee, and congratulated Piers Lauder on his work. However he suggested that the meeting organiser should be on the programme committee. CC indicated that Lauder had done all the work of the programme committee, neither he nor Ross Nealon had contributed at all. CC suggested that it would be a lot easier if the main person on the programme committee was at the local site. PW agreed. TR indicated that he would have preferred not to have had to organise the programme for the Brisbane meeting (Aug 85).

JL indicated that notices had not been appearing promptly because of newsletter problems. PW said that the call for papers had not appeared early enough.

PW had hoped to have a proceedings available at the conference, but had only received 3 papers. He has commitments from other authors, and hopes to receive 6 or 7 (additional, or total?)

CM indicated that the proceedings issue of AUUGN after the Perth meeting had been well received. It had generated several new memberships. He suggested that a proceedings issue of this conference be sent to non-members attending this meeting.

JL suggested that proceedings of meetings be published in AUUGN, and for it to be at the editor's discretion whether they are made available at the conference.

Moved (JL, seconded ?) **That the report be accepted, and PW be commended.** Carried (6-0).

11. The newsletter editor's report was not given, as the editor (Peter Ivanov) was not present. It was noted that Ivanov had resigned as editor.

    It was suggested that PW do most editing of proceedings to be published.

    JL indicated that the newsletter should not wait for papers, but should appear every 2 months regularly.

    CM indicated that the new editor should do the editing, but that printing and postage would be better if the current scheme was continued. JL indicated that decoupling the functions was a good idea.

    JC asked what should be in AUUGN, especially whether reprints of other newsletters should be included. GR suggested they should not be, JL thought they should. General opinion was that the editor should use his discretion.

    SJ suggested a newsgroup for AUUGN submissions.

12. Moved (JL, seconded TR) **That Peter Ivanov's contribution be acknowledged by the secretary.** Carried (6-0).

13. Moved (CM, seconded LS) **That John Carey be appointed newsletter editor for a period of 2 years from September 1, 1986.** Carried (6-0).

14. Moved (CM, seconded CC) **That Peter Ivanov be given a quasi membership to December 1986 as a gesture.**

15. Amendment: (JL, seconded CC) **That 1987 be substituted for 1986.** Carried (6-0)

16. Motion: carried (6-0).

    At this point TR suggested that Ivanov might be offended by this. LS agreed. There was much discussion during which several compromises were suggested.

17. Amendment: (TR, seconded LS) **That a plaque or similar token be given in lieu of membership.** Failed (1-5, TR supporting)

18. Amendment: (TR, seconded CC) **That a plaque or similar token be given in addition to membership to December 87.** Carried (6-0).

19. There is no record of a vote having been taken on the motion itself.

20. Moved (KRE, seconded CC) **That John Lions be the fourth signing officer (in addition to the president, secretary and treasurer).** Carried (6-0).

21. Moved (KRE, no seconder) **That there be no newsletter subscriptions without membership.** Without a seconder, no vote was taken.

22. Moved (CM, seconded LS) **That the fees for membership be: $50 for ordinary members, $30 for student members, and $50 for newsletter subscriptions.** Carried (6-0).

23. Moved (KRE, seconded CM) **That institutions desiring to become members may be admitted as Institutional Members upon payment of $250 per annum. This will entitle them to 2 subscriptions to AUUGN, and to send 2 representatives to AUUG meetings.**

24. Amendment: (TR, seconded LS) **That the words "at member rates" be inserted after "AUUG meetings".** Carried (6-0).

25. Motion: Carried (5-1, JL opposed).

26. Forms for use in an application for membership were presented by the secretary, for approval of the committee.

27. There was some discussion of the wording of the forms. LS moved that the words "or other paper warfare" be deleted. CC indicated that provision for payment by bankcard be added. CC suggested that provision be made for institutional members to be able to send a purchase order for payment. CM indicated that costs of international mail meant that the international air mail surplus needed to be raised to $50. No votes were taken, however all suggestions received general agreement.

28. Moved (KRE, seconded CC) **That the forms as amended be approved.** Carried (6-0).

29. Moved (CC, seconded CM) **That the next AUUG meting be held in Sydney in February 1987, and that the host be either NSWIT or Macquarie University.** Carried (6-0).

30. Moved (LS, seconded CC) **That the secretary be delegated to approach other user groups with a view to entering into co-operative relationships.** Carried (6-0).

31. Moved (CC, seconded TR) **That Lionel Singer be appointed to the budget subcommittee to replace Piers Lauder.** Carried (6-0).

It was also noted that the budget subcommittee is yet to meet, and that something should be done about this soon. The other members are CM and KENJ.

32. Moved (CC, seconded CM) **That the secretary be empowered to obtain secretarial assistance at reasonable rates.** Carried (6-0).

33. Other agenda items were deferred to the next meeting, to be held in one of the offices of LS or CC, on November 17, 1986 at 14:00.

34. Meeting closed 19:09.

Minutes of the AUUG Annual General Meeting
September 2, 1986

1. The meeting opened at 09:02. Present were an undetermined number of members of the AUUG, and several others. All AUUG committee members, except the president were present.
2. In the absence of the president, John Lions was elected chairman of the meeting.
3. Moved (Robert Elz, seconded John O'Brien) **That the required notice of the meeting be waived.** Carried without dissent.
4. No minutes of the previous AGM (Brisbane, August 1985) were available.
5. No minutes of the previous General Meeting (Perth, February 1986) were available.
6. There being no minutes, no business arose from them. John Mackin asked why there were no minutes. The secretary replied that the previous secretary didn't seem to have kept any.
7. No presidents report was given, as the president was absent.
8. The returning officer (John O'Brien) presented his report. He announced the results of the election just held for the AUUG management committee. Only 25 votes were received, or which 2 were informal. Piers Lauder withdrew from the election for President, leaving Ken McDonell to be elected unopposed (though the result would not have been altered anyway). Robert Elz was elected secretary. Chris Maltby was elected treasurer. John Lions, Piers Lauder, Tim Roper, and Greg Rose were elected to the committee, however, Lauder and Rose had withdrawn their nominations, so the next two placegetters were elected, Chris Campbell and Lionel Singer. An exhaustive preferential system was used.
9. The secretary (Robert Elz) presented his report. There were 164 ordinary members, and 48 unfinancial members still owed newsletters on Sep 1. There were 32 newsletter subscribers who were not members. 10 newsletters are exchanged with other groups, or otherwise sent without payment. There are 111 members currently unfinancial.

   The secretary then told the meeting some of the more significant results of the previous days committee meeting.

   Moved (John O'Brien, seconded Chris Campbell) **That the secretaries report be accepted.** Carried without dissent.
10. The treasurer (Chris Maltby) gave his report. Assets at June 30 were $22863.01, with approximately another $19000 outstanding, some of which has since been received. The principle outstanding debt is from Computerworld for AUUG's share of the May 1986 UnixWorld meeting.

The treasurer said that the balance sheet would be audited immediately after the meeting, and would then be published in AUUGN.

Moved (John O'Brien, seconded Russell McDonell) **That the treasurer's report be accepted.** Carried without dissent.

11. Moved (John Lions, seconded John Mackin) **That the next AGM be held in Sydney on the 26th or 27th of August 1987.** Carried without dissent.

    It was understood that the date & place are subject to variation. Moved (John O'Brien, seconded John Carey) **That discussion on chapters be deferred.** Carried without dissent.

    Glenn Huxtable stated that the Western Australian Unix systems Users Group exists, that he is president of it, and that it is functioning. He desired affiliation with AUUG but there was no great urgency involved.

12. John O'Brien attempted to move an unannounced motion, but was ruled out of order by the secretary under rule 42 of the constitution.

    The secretary had a very loud voice and was very stubborn and the motion was not heard.

13. Moved (John O'Brien, seconded Greg Rose) **That the executive committee should conduct a ballot of the members to have AUUG incorporated as a company limited by guarantee.** Carried without dissent.

    John Mackin asked for the ballot to be accompanied by a statement of the pros and cons of this change, prepared by someone competent to give such advice.

    Members of the committee agreed to do this.

14. Meeting closed 09:38.

Canberra AUUG Programme

September 1-2, 1986

Abstracts

Mike Banahan
The Instruction Set
City House
190 City Road
London, EC1
UK

Progress on Standards for Unix

This talk looks at global efforts to find a common standard for Unix and the C language. It asks why we should have standards at all, and who is arrogant enough to think that *they* know what a standard should be. It neatly avoids the hard points by making cheap jibes where it can, but reluctantly faces up to the issues occasionally. It does its best to fight for the re-introduction of hypocrisy in BNF notation.

Mike Banahan
The Instruction Set
City House
190 City Road
London, EC1
UK

What Is Unix For?

There is a lot of misunderstanding of what Unix means. Is it an operating system, an abstract definition, an implementation or a toy?

Does it matter to the users of computers in the 90s anyhow?

This talk looks at the subject from both ends, squeezes the middle and sees what dribbles out of the seams.

Paul C. Bunn
Olivetti Australia P/L
Sydney

Fault Tolerance—A Critical Issue

Hardware vendors are currently realising the demand for fault tolerance in the market place.

Olivetti, a leading supplier of computer solutions, market the CPS 32—Continuous Processing System.

The meaning of Fault Tolerance is outlined.

The market place analysed, crucial market sectors that require fault tolerance are pointed out. The method used to implement fault tolerance in Olivetti's CPS 32 is discussed.

Rich Burridge
SUN Australia P/L
Melbourne

RS-232 Port Expander for Unix

This paper describes the design of an RS232 port expander for the BSD4.2 range of Unix boxes. Using one RS232 line on the Unix machine, a cable attaches to the port expander which can have up to five RS232 devices plus one Centronics device running simultaneously.

The port expander has a 32K byte buffering capability for each channel and can therefore act as a multiple print spooler. Each channel has modem control and can therefore be used to hook up to five modems off the one RS232 port. Full XON/XOFF flow control is available on each channel as well as the trunk line.

The simple stop and wait protocol which is used is described in detail, and an overview of the port expander hardware design and software internal layout is given.

Projects are currently underway to allow the Apple Macintosh, the IBM PC, and the Commodore Amiga a multi-windowing / multi-job attachment to a BSD4.2 Unix box using this same protocol.

Frank Crawford
Jagoda Cergovska
AAEC
Lucas Heights
Sydney

A Modula-2 to C Translator

What do you do when you have some users who are addicted to Modula-2, and you're taking their computer away? Give them a new computer, right.

What if the new machine doesn't have a Modula-2 compiler? You write one.

And if you don't know the machine code (and can't find out)? You convert it to something you do know.

This paper describes a Modula-2 to C translator written for a Pyramid 90x, when the AAEC decided to do away with its PDP 11/45. It goes through some of the problems faced and conquered, the distributed processing nature of the project (some parts run on the 11/45, others on the Pyramid) and an overview of the similarities and differences between Modula-2 and C.

Ross Hand
Ortex Australia P/L
Fyshwick
ACT 2609

Unix + Guru Considered Harmful

In the traditional MSDOS market the Unix operating system is considered large and complicated. Comparisons of disk and memory requirements and the mention of over 200 standard utilities have convinced the naive users that this is true.

Unix is considered by these users as similar to the central large computer installation with its associated requirements for experts. These users have made the transition from the large central installation to the individual personal computer. The idea of exchanging a "simple" operating system like MSDOS for a complicated system is considered a backward step. This is impeding the use of the Unix operating system in the smaller, less concentrated areas of computing.

Dave Horsfall
SUN Australia P/L
20 Waltham St
Artarmon
NSW 2064

SunOS - SUN's Converged Unix System

Sun's operating system, SunOS, is the foundation on which the software is built. Based on the converged Berkeley 4.2BSD and AT&T System V, it is enhanced to provide high-performance facilities for all software packages and to maximise system throughput for workstations on heterogeneous networks.

Steve Jenkin
Softway P/L
Milsons Point
NSW 2061

Teletex - A Coming Standard

Teletex is normally presented as a TELEX replacement. It is far more than that. It is faster (min 2400 baud), error protected, independent of transmission network, CHEAP, and universal. Sounds good, eh. It is the only service where the availability of subscribers equipment is specified (free 95%). This means you get through even on busy numbers. It is defined to interwork with X.400 (universal electronic mail) and group IV FAX. Not only this, but it has to interwork with the telex network as well.

This talk will present a overview of Teletex and discuss the implementation of OTC's telex to teletex conversion facility.

Steve Jenkin
Softway P/L
Milsons Point
NSW 2061

File Systems: Unix and the Rest

Unix has a remarkable file system! It is simple, fast, and easy-to-use.

This talk will contrast various features of the Unix file system with other systems, chiefly MVS, PRIMOS, and MS-DOS.

Topics to be covered are:-

- **— logical structure of the file system**
- **— physical block layout**
- **— file types and access mechanisms**
- **— buffering**
- **— linking blocks in a file and in the free pool**
- **— outstanding features of the file systems**
- **— rough performance characteristics**

Michael Kearney
Ortex Australia P/L
Fyshwick ACT 2609

Unix SYSTEM V on the PC/AT

Three SYSTEM V's are discussed:- Xenix, Venix, and Microport V/AT. All three have different executable image formats. All three have different system call mechanisms.

SYSTEM V is only a source code standard. Who has source code for mass-market software products? The customer doesn't understand this. They are used to CP/M and MS-DOS and have been (willingly) brainwashed by slick advertisments.

Michael Kearney
Ross Hand

Ortex Australia P/L
Fyshwick ACT 2609

An Implementation of Unix on the 8086/80186 Processor

A Unix system has been implemented on an Intel 80186 processor. Despite the lack of memory management hardware only minor changes were required to the C language kernel. Two significant additions were required to the assembler code. The remaining code was simply reworked for the target processor.

The system was bootstrapped by compilation on an Altos 486 system using Xenix. The kernel was debugged using CP/M-86 running on the target hardware. Once a stable kernel was running, the Hitech C compiler was used to rebuild the system using the native hardware.

The Unix kernel was found to be remarkably robust. The kernel operated despite serious (introduced) bugs. Certain key Unix programs (Bourne shell) presented major problems due to assumptions about the underlying hardware and execution environment.

The project emphasised the need for reliable tools. Much effort was expended re-coding software to bypass compiler problems or trying to convince stupid hardware to behave reasonably.

Andrew Mack
Dept. of Comp. Sci.
University of Melbourne

4.2BSD on the ELXSI 6400

A port of 4.2BSD to the ELXSI 6400 was completed early in 1986. This presentation describes the various changes made to the kernel and user programs necessitated by the unusual architecture of the ELXSI, implementation difficulties encountered, and differences with other flavours of 4.2. The port is also compared to the System V port completed in 1984.

Ian Richards
Dept. of Inf. Tech.
CSIRO
Melbourne

A Review of Academic and Research Networking

The new CSIRO Division of Information Technology has set up a Networking Group based in Melbourne. A major objective of the group is to maximize the effective use of networking as a means of communication among researchers in Tertiary and Research institutions both within Australia and overseas. We are thus highly supportive of the existing initiatives that have resulted in ACSnet and the Division has agreed to fund the costs of incoming news traffic from overseas. However, we believe that the time is right to undertake a careful review of the networking needs of the tertiary and research community and to make recommendations for the further development of networking facilities in the future. This review is scheduled for completion during September 1986.

This presentation will, so far as is possible without pre-empting the final recommendations of the review report, discuss the issues that have emerged. In particular, the status and role of the SPEARNET initiative will be discussed and the possibility of News distribution via AUSSAT will be mentioned.

A related activity of the Division is to experiment with and demonstrate the effectiveness of new software products compliant with the emerging OSI standards. The results of any such experiements underway by the time of the presentation will be covered.

Greg Rose
Softway P/L.
Milsons Point
NSW 2061

The CCI Power/6 Computer Architecture

The Computer Consoles Inc. Power/6.32 is a new computer with an interesting architecture. The machine is very much like a VAX, but the underlying architecture is enhanced so that the low end machine performs like a DEC VAX 8600 for a price like a 750. All this using TTL technology (yes, really) with new ones coming with VLSI, ECL and other buzzwords to make it run even faster.

Obviously they did something right. Many things in fact. That is what this talk is about. Nothing secret, just why the machine is such a fast box, and why I find it interesting.

The talk includes things about the CPU, caches, and the instruction set. In particular, the differences and similarities (from a program's point of view) between the CCI and VAX are examined.

AT&T — SYSTEM V.3

Michael Selig
Olivetti Australia

An Overview of Unix System V Release 3.0

Unix System V Release 3.0 contains several new features, some of which first appeared in the interim Release 2.1. These include: Demand Paging, Record & File Locking, Remote File Sharing, Shared Libraries, and a standardized networking interface. This paper will present an overview of these enhancements, and the benefits they provide for programmers and end- users.

Colin Keith
Hugh MacKenzie
Scott Milton
John Smith
Robin Stanton
Gregory Toomey

Dept. of Comp. Sci.
ANU

Design of a Unix-based Spatial Inferencing System

We describe the design of a spatial inferencing system based on Sun workstations. The system is based around Sun workstations, using their bitmapped screens to display geographical information. Many Unix features have been used in the system including Remote Procedure Calls, the Unify database management system and Franz Lisp. We concentrate on the description of tools used and how these tools have been combined into a prototype system.

m2c: A Modula-2 to C translator

Jagoda Cergovska
and
Frank Crawford

Australian Atomic Energy Commission
Private Mailbag, Sutherland 2232

ABSTRACT

This paper describes a Modula-2 to C translator written for a Pyramid 90x, the computer which has replaced an obsolete PDP 11/45 computer at the Australian Atomic Energy Commission's Lucas Heights Research Laboratories. It describes some of the problems faced and solved, the distributed processing nature of the compiler (some parts run on the 11/45, others on the Pyramid) and gives an overview of the similarities and differences between Modula-2 and C.

PREAMBLE

What do you do when you have some users who are addicted to Modula-2, and you're taking their computer away? Give them a new computer, right.

What if the new machine doesn't have a Modula-2 compiler? You write one.

And if you don't know the machine code (and can't find out)? You convert it to something you do know.

1. INTRODUCTION

Within any computing organisation there is an accumulation of software over a number of years. When the time comes to replace a computer, users want to take across all of their favourite programs. When the Australian Atomic Energy Commission (AAEC) decided to upgrade it's PDP 11/45[1], running UNIX[2] Edition 6, to a Pyramid 90x, running OSx[3], a dual port of UNIX System V and BSD 4.2, most users found that either the programs they wanted would compile and run or there were new and better ones. One case in which this was not so was with Modula-2. At some time in the past, a Modula-2 compiler was obtained/developed for UNIX at the AAEC and a number of programs have since been written in Modula-2. As Modula-2 is not a standard part of any UNIX system, Pyramid Technology Corporation (PyrCorp) had not felt the need to develop it as part of its port of UNIX.

Because some of the Modula-2 programs were still necessary, and it was too big a task to rewrite all of them in some other language, the decision was made to develop a Modula-2 compiler for the Pyramid. This, however, raised a problem. PyrCorp is unwilling to give details of the machine code, despite the fact that they supply an assembler with the system! Accordingly, it was decided that instead of writing a compiler, we would write a translator to convert Modula-2 to C. C was chosen for a number of reasons, but primarily because it is very well supported in a UNIX environment. Also most data types and control structures can be mapped directly from Modula to C. The final reason for choosing C is its portability across systems especially, in the case of the AAEC, to an IBM 370.

1. PDP is a trademark of Digital Equipment Corporation.
2. UNIX is a trademark of AT&T Bell Laboratories.
3. OSx is a trademark of Pyramid Technology Corporation.

Other possible languages as the target of the translator were FORTRAN and Pascal. FORTRAN is so dissimilar to Modula-2 that an enormous effort would have been required. As an example FORTRAN does not support recursion, a very basic concept in Modula (and C). Although Pascal is more similar to Modula than C, Modula's extensions over Pascal would be difficult, if not impossible, to implement in Pascal.

As the translation from Modula to C is not a simple one-to-one mapping, the task was more closely related to writing a compiler than a simple translator, where the compiler's "machine code" was C rather than a more traditional machine code. Once this was realised, we had to decide between writing the complete program or making use of the Modula-2 compiler already available on the PDP. It was decided to make as much use as possible of what was available, i.e. to modify the Modula-2 compiler to suit our needs.

2. DESCRIPTION OF THE PROJECT

2.1 Overview

The project consisted of a number of parts. These were

- modifying the Modula-2 compiler on the PDP to produce output in a form suitable for further processing,
- writing the program(s) on the Pyramid to generate the C code,
- transferring the compiler to the Pyramid, and
- optimising some of the code for better performance on the Pyramid.

2.2 Modifying the Modula-2 Compiler

The Modula-2 compiler available under UNIX V6 had already been extensively modified[4] before starting this work. It was the original compiler developed by Prof. N. Wirth for an RT-11 system[5] and written in Modula-2. It was later modified by Jeffrey Tobias and others, at the AAEC and the University of New South Wales, to run under UNIX V6.

The compiler has five passes, mainly due to memory constraints, and generates code suitable for running on a PDP, but not compatible with the output from *ld*. The passes were (in order)

a. syntax analysis,

b. declaration analysis,

c. body analysis,

d. code generation for expressions, and

e. code generation for statements.

It also has separate segments for generating the symbol file and a source listing, including error messages, etc. The communication between passes is mainly achieved by intermediate files, which contain the structure of the compilation unit in a symbolic form, whereas the symbol table is kept in memory.

In simple terms, we replaced the code generation passes by code to generate C. More specifically, a modified version of the output from pass 3 was sent to the Pyramid as input to the final C generating programs (see Figure 1).

From the start, it was decided that no symbol table would be maintained by the final code generating programs; instead the Modula compiler should generate on its intermediate files information concerning

4. Actually hacked to death.

5. RT-11 is a trademark of Digital Equipment Corporation.

the data types of objects within each expression, function call, etc. This may not have been the best approach, but it made the final passes much simpler.

The generation of suitable intermediate files involved a number of major changes to the Modula compiler's symbol table. Some of these were: to keep more detailed information on the structure and order of declarations, to keep track of the scope of variable, to include back pointers to procedures and modules, to generate identifiers unique to the C code and to maintain import and export lists.

2.3 Developing the C Code Generator

The development of the C generator was an easier task under OSx than it would have been under UNIX V6, thanks mainly to the additions to the UNIX programming environment since the release of V6, and the well defined syntax of the intermediate files.

The code generator processes two streams, one for the declarations and Modula's definition modules, and the other for the body of the program. To generate C code from these two data streams, they have to be processed concurrently.

Both streams are defined in Backus-Naur Form (BNF) and, as such, are simple to parse using *yacc*. The syntax of the streams was modified from the original requirement for interpass communication to one that was more suited to translating into C.

The major problem in developing this part of the translator was to enable it to handle both streams simultaneously. As *yacc* does not process two different grammars at once, two separate programs were written, one, called *symbol*, to handle the symbol file and produce equivalent C declarations and the second, called *modula*, to process executable statements. A third program was written to merge the two outputs to form a complete C program. This was called *m2c*.

M2c runs *symbol* and *modula* concurrently and, using synchronisation characters in their output, selectively merges the two outputs.

2.4 Moving from the PDP to the Pyramid

During the early development stages, it was necessary to run the first passes on the PDP, to produce the intermediate file which was then sent to the Pyramid for further processing.

As the aim of the project was to move this compiler to the Pyramid, and as the PDP part of the compiler was written in Modula-2, once the translation process was working properly the next step was to translate the PDP parts of the compiler to C. This served two purposes; first it was a test of the translator, and second, it moved the system totally to the Pyramid.

2.5 Optimising the Code

Improvement of the code involved two distinct areas:

- removal of any size restrictions that previously plagued the parts on the PDP, and
- rewriting the C code for critical areas of the translator.

Because of the small address space available on the PDP, the initial work required many non-essential sections to be deleted from the code, for example, some error checking code. This was possible because the main aim here was to compile the compiler which can be assumed to be correct. Also the compiler uses a number of overlays for each of its passes. This adds some complication to the code and duplication of certain common routines, e.g. certain I/O routines. Much of this could be eliminated once the overlays were replaced by separate modules within the same program.

Second, as the C code generated by the translator was originally designed to be easily produced, but was not necessarily efficient, it was subsequently decided to rewrite some of the code that would be heavily used. An example of this was in the module *SysCalls*, which handles the interface between Modula and the operating system, where all the procedures were replaced by *#defines*.

3. MAJOR PROBLEMS AND THEIR SOLUTIONS

From the start of the writing of this translator, it was obvious that there were a number of differences between Modula-2 and C that would have to be dealt with. Some were overcome, others

became apparent. The major differences were

- nested procedures,
- modules and import/export lists,
- order of declarations,
- argument passing,
- set operations, and
- the *with* statement.

3.1 Nested Procedures

These are a common problem with both the Pascal and Modula-2 languages. The problem has been previously addressed for Pascal to C translation, for example by the Whitesmith's Pascal compiler, which served as a starting point.

There are two distinct problems here: the problem of unique names for identifiers within a block and the problem of the scope of objects declared external to a nested procedure but not at the global level. A nested procedure can access the variables of the procedure which encloses it, but these variables should not be accessible to other procedures at the same level as the enclosing procedure (see Figure 2).

The unique naming of identifiers was overcome by simply prefixing the actual identifier with a uniquely generated identifier for each procedure. This could cause problems for C compilers that require that identifiers be unique up to a specific (small) length, however this was not a problem in the present case as the OSx C compiler supports unlimited length identifiers[6]. The character underscore ('_') is used to separate the generated names from each other and the actual name.

The handling of scope is a much more complicated problem. Within C there are no nested procedures: all are at the same level. The first step is to unravel the nesting so that all the procedures are at the same level (with unique names). The next step is to set up the declarations so that the nested procedures can access the variables in their scope of visibility. This is done by declaring a structure of all the local variables to a procedure and passing a pointer to this structure to any nested procedure when called. This adds an extra argument to each of the nested procedures which must be treated in the same way for any further nesting. As a result, all references to local variables become structure references, with their associated overheads (see Figure 3).

3.2 Order of Declarations

Both C and Modula-2 expect all identifiers to be declared before being referenced, but in both languages there is an exception to this rule: pointers. Unfortunately, this exception is not explicitly mentioned in the definition of either language. Often it is necessary to declare a pointer to a type before the type itself is declared. This is most common in structures containing pointers either to themselves or to structures that contain pointers back to the first structure.

The problem is that the restrictions imposed by the OSx C compiler are stricter than those imposed by the local Modula compiler. The only time that C allows pointers to be defined to an undefined type, is if the base type is obviously a structure, i.e.

```
typedef struct anything *pointer;
```

whereas Modula allows any pointer to have its base type undefined.

To overcome this, it was necessary to produce the declarations for the C programs in a particular order. A check is made to ensure that any pointer type is defined before it can be used, for example

---

6. *In reality it is restricted by the maximum line length (256 chars).*
For a system that restricts the lengths of identifiers there are programs available to generate unique names (e.g. shortc from USENET's mod.sources).

within a record/structure. Given this, the order is preserved. There may be other problems with this approach, but none have been found so far.

3.3 Modules and Import/Export Lists

Within Modula, separate modules delimit the scope of identifiers and to set up "gateways" for the exchange of certain identifiers. The import and export lists are the "gateways" which specify identifiers that are either to be known to the outside world, or that are part of the outside and are to be known internally. Modules are also used to allow separate compilation within Modula.

The problem of identifiers that are entirely local is handled in the same manner as the naming of nested procedures, i.e. they are prefixed by a unique identifier. They are also declare as *static*, if appropriate (see Figures 4 and 5).

Identifiers that are to be exported for separate compilation (i.e. from *definition module*) are prefixed by the module name. Further, each *definition module* generates a file of declarations which is included by all the files that "import" any objects from this module. In effect any *import* in a module is converted to a *#include* in the corresponding C program, and the list of identifiers to be imported is ignored by *m2c* (see Figures 6, 7, and 8).

3.4 Argument Passing

Like Pascal, Modula-2 allows arguments to be passed in one of two ways, by value or by address, whereas C only allows them to be passed by value. In practice this is not generally a problem as C also provides the & operator to take the address of a variable. In most cases it is a simple book-keeping exercise to determine when to take the address of an argument and when to pass its value, especially with Modula's strong type checking. However, there are two occasions when this fails: when functions or arrays are passed as arguments. In both cases, C passes them by address rather than by value.

Within Modula, functions are passed by address as any reference to them is actually to a type which is a pointer to a function. In C this is done in exactly the same way, so there is no real problem; it is just a special case of the argument passing scheme given above.

With arrays, the problem is compounded because C treats pointer and array arguments as the same, whereas Modula distinguishes between them. In C there is no way to pass an array by value, despite the fact that structures can be passed in this way. An associated problem is that C gives no indication of the size of the array that is passed.

To overcome the problem of the size of array arguments an additional argument is passed - the size of the array, in terms of its base type. This argument has the string *_size* appended to the array name (this is an example of the power of the C preprocessor).

As C does not pass arrays by value, it has to be simulated by the translator. The first step is to allocate space to take a local copy of the array and then to copy the array into it. This space has to be freed at the end of the procedure. The appropriate code is shown in Figures 9 and 10.

3.5 Set Operations

One of the few types that could not be directly mapped into C was *sets*. The major problem with a *set* is not the type, as this can be declared as an *unsigned int*, but the operations on *sets*. Modula defines the normal operators such as '+', '-', '*' and '/' to perform specific operations on *sets*. Modula's relational operators also can be applied.

TABLE 1. *Modula-2 Set Operations and Relations and the Corresponding C Code*

Operation	*Symbol*	*C Code*
Set Union	+	*(s1) | (s2)*
Set Difference	-	*(s1) & !(s2)*
Set Intersection	*	*(s1) & (s2)*
Symmetric Set Difference	/	*(s1) ^ (s2)*

Relation	*Symbol*	*C Code*
Equality	=	*(s1) == (s2)*
Inequality	#	*(s1) != (s2)*
Set Inclusion	<=	*((s1) & (s2)) == (s1)*
	>=	*((s1) & (s2)) == (s2)*
Set Membership	*IN*	*((1 << (x)) & (s1)) != 0*

Once the problem of distinguishing between operations on Sets and other types was solved it was relatively simple to use bit manipulation to perform the required operation. Table 1 shows the set operators with their corresponding C code.

3.6 The **with** *Statement*

Modula-2 has a statement not available within C; this is the *with* clause. Its effect is to qualify a record so that its field identifiers may occur within the statement sequence by themselves, i.e. without being prefixed by the record identifier and the period. For example

```
WITH d1 DO
  day := 10; mo := Sep; yr := 1986
END
```

is equivalent to

```
d1.day := 10; d1.mo := Sep; d1.yr := 1986
```

There is no way this can be duplicated within C, however a close approximation can be obtained by declaring a local variable as a pointer to the appropriate structure and initialising it with the address of the specified variable. Then, a reference to any field is replaced by a pointer reference. The above becomes

```
{
    register    Date * _FL1 = &d1;

    _FL1 -> day = 10;
    _FL1 -> mo = Sep;
    _FL1 -> yr = 1986;
}
```

4. UNSUPPORTED FEATURES

Although the translator supports most of the features of Modula-2, there are two that have not been implemented:

- Concurrent processes. These could be implemented, but, as it was not needed for the applications we were interested in, it was not worth the effort. Further something similar could be done using the existing system calls *fork* and *exec*.
- Priorities and monitors. These don't really make sense under UNIX as it is designed for low level handling of interrupts.

5. FUTURE DIRECTIONS

Now that the translator is working, the task has changed to one of improving it, both in terms of the code produced and its internal operation. As mentioned previously, the first task is to remove as much redundant code as possible. This is both possible and simple as now that the previous restrictions on the size of the code have been lifted, the original passes of the Modula compiler can be examined as a single unit rather than as separate units.

The original Modula-2 compiler, and this translator, were based on Wirth's first definition (Wirth 1982[7]). Since then, he has published two more (Wirth 1983[8] and Wirth 1985[9]). As each edition has introduced small changes to the language, a future task is to bring our translator up to date[10].

One task that would be useful would be to modify the code generating passes to conform with the requirements of PyrCorp's Common Language Environment (CLE). This was not possible when the project started as CLE had not been implemented by the Corporation at that time.

6. CONCLUSIONS

The Modula-2 to C translator, *m2c*, is now running on the AAEC's Pyramid computer and is being used for its original purpose, that of moving programs from the aging PDP 11/45. Even though it is now in general use it is sure to have a number of bugs and, as indicated above, there is probably more development work to be done.

The project has demonstrated the closeness in structure of Modula-2 and C. Despite the fact that they use different symbols for operators and control structures, the difference is merely cosmetic, the underlying fabric being the same.

---

7. *Wirth, Niklaus, "Programming in Modula-2", Springer-Verlag, 1982.*
8. *Wirth, Niklaus, "Programming in Modula-2", Springer-Verlag, Second Ed., 1983.*
9. *Wirth, Niklaus, "Programming in Modula-2", Springer-Verlag, Third Ed., 1985.*
10. *If we can keep up!*

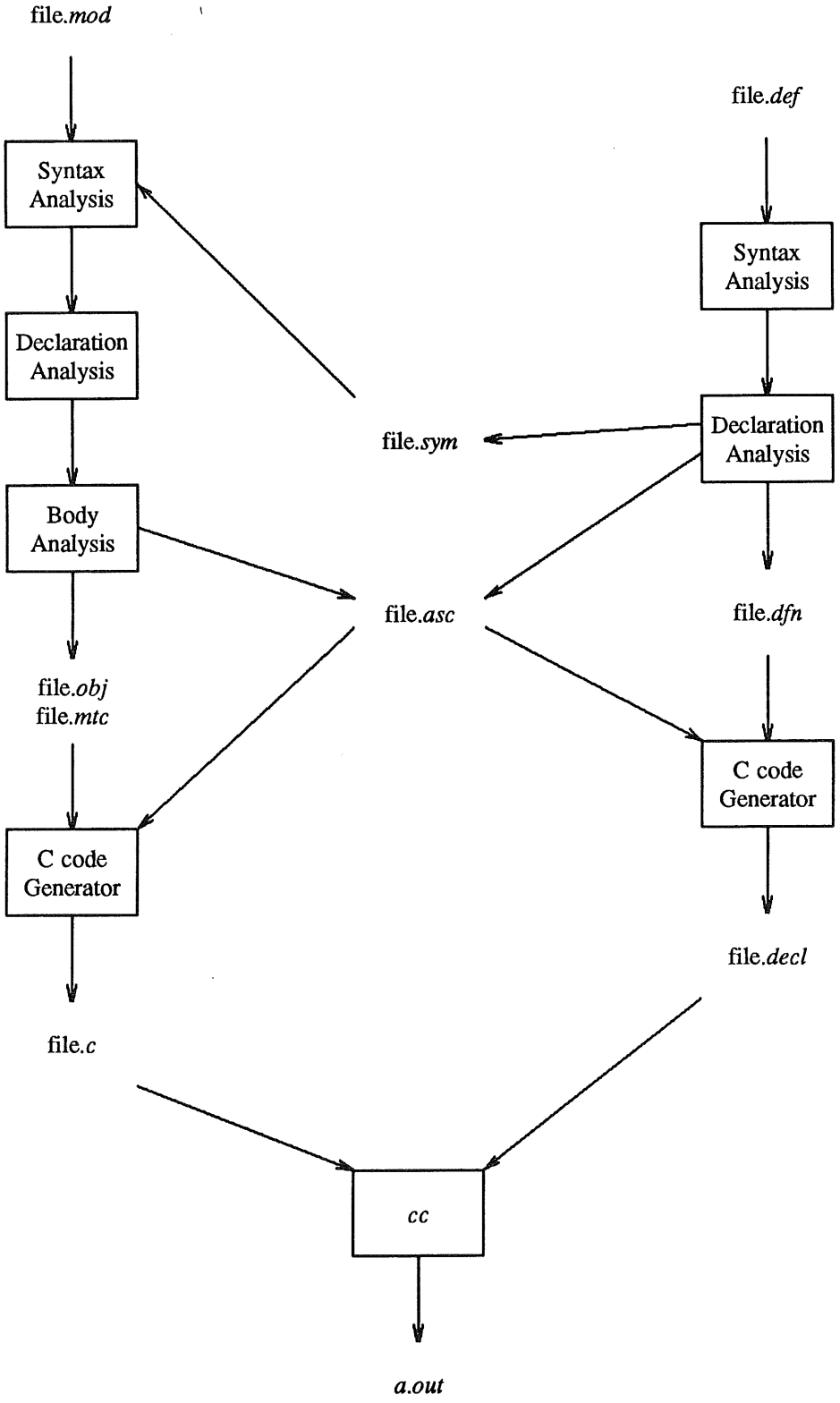


Figure 1. *Compiler Overview*

```
PROCEDURE Primes;
  VAR
     i: INTEGER;
     prime: INTEGER;

  PROCEDURE Reset(k : INTEGER);

  BEGIN
    WHILE k < size DO
      flags[k] := FALSE;
      k := k + prime;
    END;
  END Reset;

BEGIN

  FOR i := 0 TO size DO
    IF flags[i] THEN
      prime := i + i + 3;
      Reset(i + prime);
      count := count + 1;
    END;
  END;
END Primes;
```

Figure 2. *Example of Nested Procedures within Modula-2*

```
/* TYPES */
typedef struct _1 {
    INTEGER i;
    INTEGER prime;
}
                    _1;
typedef struct _2 {
    _1 * _P;
    INTEGER k;
}
                    _2;

static
        void _1_Reset (k, _P)
        INTEGER k;
_1 * _P;
{
/* VARS */
    _2 _A;

    _A._P = _P;
    _A.k = k;
    while (_A.k < size) {
      flags[_A.k] = FALSE;
      _A.k = _A.k + _A._P -> prime;
    }
}


static
        void Primes () {
/* VARS */
            _1 _A;

    for (_A.i = 0; _A.i <= size; _A.i++) {
      if (flags[_A.i]) {
          _A.prime = _A.i + _A.i + 3;
          _1_Reset (_A.i + _A.prime, &_A);
          count = count + 1;
      }
    }
}
```

Figure 3. *Corresponding C Code for Nested Procedures*

```
MODULE MainProgram;

  VAR
    i: INTEGER;

  MODULE RandomNum;

    EXPORT Random;

    VAR
      Seed: INTEGER;

    PROCEDURE Random(): INTEGER;
      CONST
      Modulus = 7415;
      Inc     = 25543;

    BEGIN
      Seed := (Seed + Inc) MOD Modulus;
      RETURN Seed;
    END Random;

  BEGIN
    Seed := 0;
  END RandomNum;

BEGIN
  i := Random();
END MainProgram.
```

Figure 4. *Example of an Internal Module*

```
/* IMPORTS */
#include "SYSTEM.decl"

/* VARS */
static INTEGER i;
static INTEGER _1_Seed;

/* Module RandomNum */
/* EXPORTS */
/* _1_Random */


/* CONSTS */
#define _1__2_Modulus   7415  /* type = INT-CARD */
#define _1__2_Inc 25543 /* type = INT-CARD */

static
        INTEGER _1_Random () {

            _1_Seed = (_1_Seed + _1__2_Inc) % _1__2_Modulus;
    return (_1_Seed);
}

static
        void RandomNum () {

            _1_Seed = 0;
}

/* End RandomNum */

/* End module RandomNum */
void MainProgram () {
    static int  _Done = 0;

    if (_Done++)
      return;
    RandomNum ();
    i = _1_Random ();
}

/* End MainProgram */

main (argc, argv)
int     argc;
char  **argv;
{
    Program_Name = *argv;
    ArgCount = argc;
    ArgVector = argv;
    MainProgram ();
}
```

Figure 5. *Corresponding C Code for an Internal Module*

```
DEFINITION MODULE RandomNum;
  EXPORT QUALIFIED Random;

  PROCEDURE Random(): INTEGER;

END RandomNum.
```

```
IMPLEMENTATION MODULE RandomNum;

  VAR
    Seed: INTEGER;

  PROCEDURE Random(): INTEGER;
    CONST
      Modulus = 7415;
      Inc     = 25543;

  BEGIN
    Seed := (Seed + Inc) MOD Modulus;
    RETURN Seed;
  END Random;

BEGIN
  Seed := 0;
END RandomNum.
```

```
MODULE MainProgram;
  FROM RandomNum IMPORT Random;

  VAR
    i: INTEGER;

BEGIN
  i := Random();
END MainProgram.
```

Figure 6. *Definition, Implementation and Program Modules*

```
...

#ifndef     _RandomNum
#define     _RandomNum
/* IMPORTS */
#include "SYSTEM.decl"

/* EXPORTS */
/* RandomNum_Random */


INTEGER RandomNum_Random (  ) ;
void RandomNum (  ) ;
#endif
/* End of symbol */
```

```
...

/* IMPORTS */
#include "SYSTEM.decl"
#include "RandomNum.decl"

/* EXPORTS */
/* RandomNum_Random */

/* VARS */
INTEGER Seed;

/* CONSTS */
#define _1_Modulus      7415  /* type = INT-CARD */
#define _1_Inc    25543       /* type = INT-CARD */

INTEGER RandomNum_Random () {

    Seed = (Seed + _1_Inc) % _1_Modulus;
    return (Seed);
}

void RandomNum () {
    static int  _Done = 0;

    if (_Done++)
      return;
    Seed = 0;
}

...
```

Figure 7. *Corresponding C Code for Implementation and Definition Modules*

```
/* Some necessary type definitions */
typedef int INTEGER;
typedef unsigned int    CARDINAL;
typedef unsigned int    BITSET;
typedef enum {
    FALSE, TRUE
} BOOLEAN;
typedef char    CHAR;
typedef double  REAL;

#include "SM.decl"
unsigned int    ArgCount;
char  **ArgVector;
char   *Program_Name;
/* MODULE MainProgram */
/* ModuleKey = 0xacbe, 0x30, 0xa, Name = MainProgram */

/* IMPORTS */
#include "SYSTEM.decl"
#include "RandomNum.decl"

/* EXPORTS */

/* VARS */
INTEGER i;

void MainProgram () {
    static int  _Done = 0;

    if (_Done++)
      return;
    RandomNum ();
    i = RandomNum_Random ();
}

/* End MainProgram */

main (argc, argv)
int     argc;
char  **argv;
{
    Program_Name = *argv;
    ArgCount = argc;
    ArgVector = argv;
    MainProgram ();
}
```

Figure 8. *Corresponding C Code for the Program Module*

```
MODULE Sum;

  VAR
    data : ARRAY [1 .. 20] OF INTEGER;
    value: INTEGER;

  PROCEDURE sum(x: ARRAY OF INTEGER): INTEGER;
    VAR
      i: INTEGER;
      s: INTEGER;

  BEGIN
    s := 0;
    FOR i := 1 TO HIGH(x) DO
      s := s + x[i]
    END;

    RETURN s
  END sum;

BEGIN
  value := sum(data)
END Sum.
```

Figure 9. *Example of Argument Passing in Modula-2*

```
...

/* VARS */
INTEGER data[20 /* 1 .. 20 */ ];
INTEGER value;

/* TYPES */
typedef struct _1 {
    INTEGER * x;
    CARDINAL SM_HIGH (x);
    INTEGER i;
    INTEGER s;
}
                    _1;

static
        INTEGER sum (_APB0_, SM_HIGH (x))
        INTEGER * _APB0_;
CARDINAL SM_HIGH (x);
{
/* Array arguments */
    INTEGER * x;
/* VARS */
    _1 _A;

    x = (INTEGER *) Malloc ((SM_HIGH (x) + 1) * sizeof (INTEGER));
    (void) bcopy (_APB0_, x, (SM_HIGH (x) + 1) * sizeof (INTEGER));
    _A.x = x;
    _A.SM_HIGH (x) = SM_HIGH (x);
    _A.s = 0;
    for (_A.i = 1; _A.i <= SM_HIGH (_A.x); _A.i++) {
      _A.s = _A.s + _A.x[_A.i];
    }
    (void) free (x);
    return (_A.s);
}


void Sum () {
    static int  _Done = 0;

    if (_Done++)
      return;
    value = sum (data, 19);
}

...
```

Figure 10. *Corresponding C Code for Argument Passing*

A UNIX† Implementation on the Intel 80186 processor

Michael S. Kearney

Ross J Hand

Ortex Australia

ABSTRACT

This paper describes the implementation of a Unix L7 (Ausam) system on an Intel 80186 processor. The description is intended as an informative explanation of the approach used and the sort of problems encountered, rather than a detailed technical exposition.

How is the Unix system 'ported' ?

Since almost all of the software that makes up a 'Unix system' is written in the language C, it would seem a relative straight forward exercise to make this software available on a new processor. Obviously a C compiler for the new processor is required and many ports of Unix have begun with the writing of this compiler. This is a major task in itself and is not discussed here.

The minimum resources needed to carry out a port would be:

1. A host machine on which the system software will be compiled and a target machine on which the result of the compilation will actually run. A Unix system for the host is almost mandatory.

2. A cross compiler compatible with the host and target. This compiler must run on the host system, yet produce code for the target system. If this compiler cannot eventually be built to run on the target, then a native compiler for the target is required.

3. A loading mechanism. The output from the cross compiler must transported to the target machine by some means. This could be magnetic tape, random access disks (floppy or hard), or a serial data link. The random access devices are the most flexible since ideally, a Unix file system can be constructed on the host, ready for the target. The other alternatives are viable, but need additional effort to create bootstrapping programs that will create the filesystem on the target hardwares disk system.

4. Some debugging tools on the target. These may range from hardware logic analyzers, monitor programs running on the target, to a simple operating system and binary debugger co-resident with the new Unix system on the target. As will be seen later these tools are probably the most important pre-requisite.

5. Finally you need the source of the system.

With these resource assembled the porting can proceed.

† UNIX is a trademark of Bell Laboratories.

In our case the host and the target were of the same processor type. An Altos 586 (an 8086 based machine) running Xenix 3 was the host, and the target was an Ortex Net-186 (an 80816 based machine). The compiler was the Xenix 3 compiler. We had no source for the Xenix system. The kernel source was for the PDP-11.

The 8086 and 80186 processor are essentially identical. The 80186 has improved micro-code and integrates a number of necessary support devices into a single silicon chip. These additional devices are 3 timers, 4 interrupt lines, 2 DMA controllers and 8 programmable chip select lines for external use.

The loading mechanism was a 5 1/4" floppy disc. This disc format was common to both the host and the target. The floppy had a Unix file system from the host and it was intended that the target bootstrap itself from this disk. The Net-186 had a ROM-based monitor with a small set of memory display and modify commands. It could also load the first block from the floppy and execute it.

Memory management

With the necessary hardware assembled the next step in the porting process is to decide how to implement the machine dependent features of Unix on the target hardware. Memory management is the principal problem.

The 8086 and the PDP-11 are similar in architecture. Both use a 16 bit word and support separation of code and data spaces. However, the PDP-11 has a memory management and protection system, whilst the 8086 has none. In order to allow the 8086 to expand its address space, the processor supports segmentation registers. These 16 bit registers are multiplied by 16 (implemented as a shift of 4 bits) and added to the 16 bit memory offset to form a 20 bit effective address. This allows the processor to address 1 Mbyte of RAM. Segmentation registers are provided for code, data, stack and an 'extra' space memory references. In practice only the code and data space registers are generally useful, since for the C language model the stack and data segment values must be the same. The 'extra' or ES segmentation register is effectively useless except for assembler coded routines (Intel designers please take note).

By manipulating the values in these segment registers, it is possible to achieve rudimentary memory mapping functions. Importantly, no protection exists, since the processor has only a 'privileged' mode.

Finally, in the target system all peripheral devices were accessed via specific port instructions. This requires that C language code call a small assembler routine to access peripheral registers rather than direct memory references using pointers.

The only serious memory management problem to overcome is the mapping of the 'user area'. This is a kernel data structure that holds all per-process information not required by the kernel until the process is actually running. Space is provided in the 'user area' for the kernel stack, which is switched with the processes as they are run on the cpu.

In the PDP implementation, this data structure is located in physical memory just below the memory allocated to the process image. By using the memory management hardware of the PDP, when the process is running, the data structure is mapped to appear in the top 1K of the kernel logical address space. This is not possible using the 8086 segmentation registers.

The method we choose is to maintain two copies of this data structure: one located in physical memory beneath the process image, and the other located in the top 1K of the kernel data space. As the kernel switches between processes, the active data maintained by the kernel is copied back to the base of the process image for the departing process, and a fresh data copied from the arising process into the kernel data space. This change was relatively easy to implement as context switching is localised into two kernel routines 'save' and 'resume'. Both of these are implemented in assembler code.

Implementation

Having worked out how we would eventual implement the required functions on the 8086, we began with the porting.

We worked our way through the various software needed as the system is bootstrapped.
This was:

1. the primary bootstrap
2. the secondary bootstrap
3. the Unix kernel
4. the 'init' process
5. the shell

Rather than recount the detail of this work a small example will suffice. When an interrupt occurs (either software or hardware) it may be necessary to switch stacks from the user stack in user memory, to the kernel stack in the kernel memory. In the PDP this stack switch is carried out be the hardware. In the 8086 the previous code segment and instruction pointer are pushed onto the stack by the hardware before the interrupt routine starts. It is necessary at this point to determine whether a new stack is required. If the interrupt came from a user program, then a switch must be made to the kernel stack. If the interrupt came from the kernel, then no switch is required. Making this choice requires program code and machine registers, so the old values must be preserved. For re-entrant programming, they are pushed onto the current stack. Since the old values are needed by the kernel, once the correct stack has been established, it is necessary to copy these register from the previous stack onto the new one. This may seem confusing, and it certainly was to us initially. This confusion was an ongoing experience (quite common in software ?).

Debugging

Debugging code that switches stacks can be difficult. In our case, putting "printf's" in the code was not really efficient. Neither was starring blindly at page of assembler code that is wildly copying registers from one stack to another.

As part of other work, we had a CPM-86 system that ran on the target machine. This system did not use interrupts and had a reasonable machine code debugger (DDT-86). We configured a CPM system to occupy the top 128K of RAM, and restricted the automatic memory sizing of Unix so as to avoid using this memory. By placing a "halt" instruction in the Unix start up, we could load the Unix system normally and it immediately halted. By then loading CPM into high memory, and using this system to load the debugger DDT-86, we then re-started the Unix system but under control of the debugger. The problem code could then be single stepped, breakpointed etc.

When using this technique no symbolic addresses were displayed by the debugger on the target machine. To overcome this lack of symbols three terminals were used as a "window" system. Two were attached to the host, one displaying the source code, another running "adb" on the constructed kernel. The third was attached to the target. The source was examined to choose a break point. Adb was used to locate the address and disassemble the code in the region. Finally this address was used to set a breakpoint on the target. This method was not just amazing, it was amazingly amazing !

Running native (A bit like going native ?)

After much effort we had a running Unix system. Actually it was a very poor system - staggering would be a better word ! Only the clock was interrupt driven. The disk and terminal device drivers simply polled the hardware until completion was seen. From this base the system was developed by adding interrupts, improving device drivers and fixing bugs that appeared.

The major remaining task was to implement a C compiler on the target machine, and allow the system to rebuild itself on the target. We obtained the source of the Hitech C compiler from Clyde Smith-Stubbs. This compiler was chosen because:

1 It was available in source at a reasonable price
2 It produced good code
3 It was a local product with excellent support

Unfortunately, this compiler had a long integer order that was different from the Xenix compiler (the Xenix compiler was wrong !). Briefly, this means that systems produced using the Hitech compiler could not operate with Xenix filesystems. However, all development was still being performed on the

Xenix system. Filesystem conversion programs that made the necessary byte/word swaps were written to overcome this.

A more serious problem was that programs built with the Xenix compiler were incompatible with the Hitech kernel if any system call passed a long value. Again, this was temporarily solved by creating a special system call library that word swapped all long values.

Observations

The system is now self-compiling on the Net-186 target. A large number of the standard Unix programs have been built. The performance of the system is subjectively better than a DEC 11/23. The "dc" benchmark runs in 20 seconds. This is a poor reflection, since the compiler could not optimise 'dc'.

The system has drivers for a range of hard disk types, a streaming cartridge tape, parallel printer and serial terminals. A floating point emulator is built into the kernel, since the hardware does not provide an 8087. The emulator is very slow !

An important improvement in the system performance was achieved by rationalising the saving and restoring of the user data area. Our first solution was to copy this area back to user memory whenever the 'save' routine was called. This occurs frequently, at least once every system call. Since the copy required 2 milli-seconds, the system performance was poor (!). By delaying the "outward" copy until a context switch was in progress, (in the 'resume' function), the performance loss due the copying of this data structure was dramatically reduced.

When programs execute, they are allocated sufficient space for the code, and 64K for the data/stack. This tends to use memory very quickly, but minimises the risk of processes interfering with each other. An alternative execution model is supported to reduce memory usage. This allocates a fixed stack (fixed at link time) below the data. When the process executes, space is only allocated for the stack, data and bss as given in the program header. If the data space now needs to grow more memory can be allocated directly above the process, or the process can be swapped if the required memory is already in use. Certain programs have excessive stack requirements and cannot use this execution model.

A large memory user was the shell (Bourne) since it is active for every terminal. Attempts to run with the fixed stack model failed. Investigation revealed that it relies on memory fault signals to increase its data space. This is impossible to provide on an 8086. The Bourne shell was the only program we encountered that relied on specific machine features. As a passing comment it seems to us that the data allocation scheme used by the Bourne shell could be improved (re-written !).

In order to provide a wider base of application software, the system provides the capability to run Venix-86 programs. The Venix system call uses a different software interrupt to access the kernel, and has minor differences in its system calls. The mapping is carried out by having a different 'sysent' table for Venix system calls, and providing a flag in the u-area to indicate the source (either Unix or Venix) of the current system call. Since Venix expects a System V environment, the tty ioctl call supports both L7 and System V definitions. The Venix RM-Cobol package was used to validate this enhancement. Other software from Venix (notable the C compiler and a Basic interpreter) function correctly.

A large effort in the work was dealing with poor peripheral hardware. On a machine where memory is consumed quickly and is limited, swapping occurs regularly. Our initial disk driver could only perform a single block transfer. Multi-block transfers were simulated by the driver. The use of DMA allowed multi-block transfers, but the improvement was marginal. This was due to poor controller design. To be fair, the controller was probably targetted at a single user system where its performance would have been adequate. We observe that many PC based Unix system have similar peripheral performance bottle necks. This is a pity, since alternative (faster) hardware is available.

Future

The measure of a programmer is their ability to master a significant program. From our experience, we would recommend the task of porting the Unix system to new hardware as a intellectually rewarding task.

From a commercial view, our product is too little, too late. The days of the 8086/80186 are numbered. For the size of the potential market in Australia for this product, the costs involved are prohibitive: $US 66,000 for Sys V R3, $US 29,000 for binary distribution. This is approximately $A 160,000. With alternative systems (Xenix, Venix, Microport) being marketed at $A 1,000, you need a big volume to even break even.

It would appear that Australian expertise will be reduced to installing binary systems from overseas, or giving lectures on how to use Xenix. This is a waste. (I think you ought to know I'm feeling very depressed ...)

File Systems
UNIX and "the Rest"

W. S. Jenkin

Softway Pty Ltd

ABSTRACT

UNIX has a remarkable file system! It is simple, fast, and easy-to-use.
This talk contrasts features of the UNIX file system with other systems, MVS, MS-DOS, RSX-11M, and PRIMOS.

Topics covered are:-
— logical file system structure
— physical block layout
— file types and access mechanisms
— buffering
— linking blocks in a file and in the free pool
— outstanding features and problems of the file systems

1. Introduction

The UNIX[1] file system is very good at what it was designed for – time-sharing. So how does it compare to other, differently targeted, file systems? I've had professional experience with IBM MVS and DOS/VS, MS-DOS, RSX-11M, and PRIMOS, so can give a comparative analysis of UNIX and those systems.

Well, what are the good points of the UNIX file system, and what are its problems? UNIX hides the underlying hardware from the user – this is its best point and worst. There are a lot of plus's:

— All the code to handle a device sits in one spot – the driver.

— Users see just one sort of file – a stream of bytes.

— On top of a UNIX file system any access method or file characteristic can then be emulated.

— Lastly, nearly any file naming or security mechanism can be emulated. (At Bell Labs "V8" allows switchable file systems, which really does allow almost any scheme to be (transparently) part of a UNIX file system).

---

1. UNIX is a trademark of AT&T Bell Laboratories

The UNIX kernel has a very efficient interface to the file system resulting in very good performance in multi-tasking time sharing.

1.1 Problems

Having the 'real world' hidden from you means you can't access it when you need to – say for writing specialised (high performance) systems (V8's file system switch could solve this). More seriously, there is performance degradation and file system corruption to worry about.

1.2 Performance Degradation

The way UNIX keeps its free chain, just released blocks are the next to be allocated, and no effort is made to sort the whole free list. This initially gives optimally spread files, then slowly degrades into a mess with files fragmented mercilessly over the disk. Solutions: Every day run "fsck –sX" to rebuild the free list so fragmentation is slowed. Periodically, rebuild the entire file system from your backups. This not only optimises all file allocations, but gives you confidence in your backups. (DON'T do this with only 1 copy of your data.... you run the risk of losing everything).

1.3 Corruptions

Walk up to a busy UNIX system and turn off the power! (Late news: some UNIX systems can survive this.) Because disk writes are asynchronous, an operational file system is mostly inconsistent. Fsck can be used to repair it, but data will be probably be lost. Reliable file systems have been built on top of UNIX – but performance is traded for reliability. Given the MTBF of hardware and power supplies, this is rarely a problem. In fact, very few systems recover from random power hits.

1.4 Problems with Berkley's 4.2BSD

Berkley 4.2 offers a 'fast' file system. Yes, it does mostly go more quickly, at the cost of being much more complex and having a new set of problems. 4.2 BSD uses large blocks (4–8k vs 1k for sys V.2) organised into 'cylinder groups' with a bit map for the free list. Block allocation strategies are complex – it tries to always place files optimally. Under extreme conditions, approaching 100% disk full, it fails. The system spends all its time trying to 'do the right thing'. System disk accesses skyrocket and user throughput slumps. 4.2BSD has a solution – disk full, for non-super users, is defined as 90% disk space – out of the danger zone. Only super users can force the system into the quagmire. The other major cost of 4.2BSD is kernel space for buffers – memory is not yet *that* cheap and plentiful. Each buffer uses 4-8 times that used in system V - so it's fewer buffers, or bags more memory, or both, for you. Raw disk utilisation is increased, and if that wasn't your bottleneck, you lose out badly. (In time sharing systems, most tasks are quite small, hence most disk I/O is for small lumps of data – ideally file systems could be 'tuned' so disk I/O size always matched the block size. Preferably, use big or small blocks in exactly those places they are needed. This *isn't* big file == big block size, but long I/Os == big block size.)

2. File System Descriptors

There are 4 structures and 1 action any file system must provide. Structures:

— Directory

  — File Naming

  — Logical File Structure

  — Access Controls and Permissions (Read, Write, Execute/Search, Delete, etc)

— Free List

— Internal File Linkage

— Data Layout of Files

Action:

— Physical I/O through Buffers

3. 'The REST' – Descriptions

3.1 IBM – MVS

IBM, the world's best marketeers, lead off.

MVS can be thought of as a superset of DOS/VS. DOS misses out on catalogs, PDSs, and automatic area allocation. It won't be talked about again here.

3.1.1 Concepts

Files consist of up to 8 "extents" of contiguous tracks or cylinders on a DASD (Direct Access Storage Device). Files start on a track or cylinder boundary. Files are preallocated with a 'primary' extent and grow in increments of the 'secondary' extent. Files appear to the user as a stream of blocks – each block filled with contiguous records, of fixed or variable lengths.

Blocks are stored on disk in 'key, count, data' format and are separated by 'inter-record gaps'. Records and blocks can each be up to 32k long. The 'key' field is optional and used by the hardware to support ISAM(Indexed Sequential Access Method). 'Count' is the numeric identifier for the block. The hardware matches this field with the block number specified in the channel program. The 'data' field is just that. As well as these there are lengths for the header and data fields. On the new FBA (Fixed Block Architecture, don't you just love acronyms!) devices, tracks are divided into 32 byte sectors. Blocks start on sector boundaries and have a trailing 12 byte CRC as well as the other overhead of 384 bytes per block. The capacity of the 3375 drive using maximum and minimum sized blocks is 409Mb and 31Mb respectively.

File names and extents are stored in VTOCs (Volume Table Of Contents, a cylinder at the 'end' of the disk). File names need only be unique within the one volume. VTOC

entries look suspiciously like card images and have record names like 'FORMAT' and 'HEADER'. File names can be up to 44 alphanumeric characters long, but must be broken by '.'s into strings (or levels) of 1 to 8 characters. MVS adds another few levels of complexity – catalogs and Partition DataSets (PDSs). Catalogs are VSAM files (last one! Virtual Storage Access Method :- a B-tree replacement for ISAM) that serve to export file names from a volume to the rest of the system. Partition DataSets contain aggregates of (usually text or code) files, all with the same record and block sizes. It has a directory that contains member names, and their relative start block within the file, and the member size. Members are contiguous blocks within the file, and start on block boundaries. Space within a PDS is not automatically reused. It is reclaimed by the user 'compressing' the file. This usually takes a while, and probably is done just after a user has tried to save a member being edited and got the informative message "SC037 abend".

Access methods and that's the end. There's SAM, BSAM, QSAM, VSAM, ISAM, QISAM, DAM, and BDAM'd files. The access method MUST be specified when creating a file. VSAM is very flexible and efficient, but of course uses completely different (and incompatible) utilities to the others.

Data bases are another story and usually consist of a set of DAM files.

Enough jargon , here is the translation.

3.1.2 Directory

— This is multipart. Unique file names are always enforced within a volume. If the file name is exported (cataloged), it is unique across all volumes.

— The name space is flat, with names up to 44 (alphanumeric) characters, restricted to 1-8 characters per level.

— There are NO inherent access controls or permissions. RACF, or similar, must be purchased for this.

3.1.3 Free List

— Unused extents are noted in each VTOC.

3.1.4 File Linkage

— For files, datasets are stored as up to 8 sets of contiguous tracks or cylinders. This is stored in the VTOC. For Partition DataSets, members are stored as contiguous blocks. This is stored in the PDS directory. For both, only start address and length are required.

3.1.5 Data Layout

— Files are contiguous and start on track/cylinder boundaries.

— PDS members are contiguous blocks starting on block boundaries.

3.1.6 Buffering

— All buffers are allocated and handled by the user. IBM supplies routines to handle each flavour of I/O for each device type. Each program must link these in. (Code is not shared, so as well as storing 'n' copies of these in lots of code libraries, they take up lots of room when executing.) In addition, physical I/O (necessary for good performance) requires these routines to 'know' all about the drives being used. These routines call the operating system drivers. So new drives cause lots of problems. Minimally, everything has to be relinked, if not recompiled, to use new block sizes to be efficient on the new drives.

3.1.7 Utilities

— Each access method has its own set of utilities. Each of these is incompatible and very different. (eg:- REPRO copies VSAM files, IEBCOPY copies PDSs, IEBGENR copies SAM and DAM files, and databases need their own special ones...)

— Each subystem, batch or TSO, have very different facilities and command languages. It is actually *possible* to get limited access to the other subsystems facilities. It usually isn't worth the trouble.

3.1.8 Missing

— System wide buffering and caching of 'popular' files.

— Automatic reuse of unused space.

— Fragmented free space has to be collected occasionally. Both for entire volumes and Partition DataSets.

— Elastic file sizes.

— Resource usage limits (Go till it fills!).

— Access controls and permissions.

— File compression utilities.

— Universal text format (with space compression).

— Universal file handling utilities.

3.1.9 Excels at

— Bulk file transfers: 1Gb backups in 15-20 minutes!

— Really good at stand alone manipulation of lots of data.

3.2 MS-DOS.

MS-DOS is only meant to be a single-user, single-tasking system for micros. It certainly shows its heritage, but is worth considering here.

MS-DOS has real hierarchical directories, per logical drive. Files are streams of bytes that can be appended/truncated at will. Unused space is automatically reclaimed. All

available blocks on a drive can be used. Very UNIX like! The FAT (File Allocation Table) makes all this possible. Each of the 8 logical drives (A: – H:), has its own FAT with a maximum of 4096 entries, each 12 bits long. Each entry represents a 'cluster' of sectors on disk. Cluster sizes range from 128 bytes to 16k bytes, doubling at each step. The maximum logical drive size is 32Mb. Each FAT entry is a pointer to the next cluster in this chain. The end of a chain is marked by 'FFF'. Free blocks, as well as normal files are chained together this way. The (loadable) device drivers hide the physical facts from the user. A logical drive can be a floppy, part of a hard disk, or be in main memory (a 'RAM' drive). Cluster, directory, and drive sizes are static and set when a disk is formatted. To change them means wiping all the data on a disk. This is not to be done lightly (or quickly). Buffering is handled transparently by the system – there are start up parameters specifying the number of file 'handles' and buffers allowed.

Using 'RAM' disks speed system performance, but bites savagely into the 640k total memory space available.

3.2.1 Problems

— Directories seem to be fixed sizes, not elastic. So inspired guesswork is needed when originally formatting the disk. (At least info from deleted files is left around so they can be reclaimed.)

— Clusters get 'lost' occasionally. They are not in the free chain or any file. The program 'chkdsk' is provided to sort this out. It does a reasonable job.

— The 'Norton Utilities' are a must. They provide all sorts of facilities to patch up disk problems (like getting back a deleted file).

3.2.2 Missing

— Access control is totally missing.

— Permissions are rudimentary.

— There is no concept of "file owner". (Expected in a single user system.)

— Clustering wastes heaps of space. A 20Mb disk uses 8k clusters! (For 20 short command files, this is 157k lost.)

— Files can get fragmented – regular backups and reloads are needed to avoid performance degradation. (On a micro with a 20Mb hard disk and only 360k floppies, this is quite a chore and rarely done.)

3.2.3 Really Good Bits

— It is very surprising such a simple system performs so well and with such good facilities.

— It's simple, effective, and very widely used.

— It provides heaps of utility and hides the underlying devices from the user, and mostly allocates files optimally.

3.3 RSX-11M and VMS

VMS is a 32-bit extension of RSX-11. It does heaps more and contains a more sophisticated file system (read hierarchical directories with permissions). The VMS file system is based on RSX-11, so although only RSX-11 will be discussed here, most of it can be applied to VMS.

RSX-11 is multi-user, multi-tasking. It understands 16-bit words and octal *really* well. It is amazing how much can be packed into 16 bits!

Yes Virginia, it does have directories and mostly reasonable file names. By using a kludge, RAD50(Radix Octal 50), 3 characters (of 0-9, A-Z, '.', and $) can be stuffed into 16 bits. File names are 9 characters long, with an optional 3 char suffix. Every file has a version number as well, it is 16 bits printed in octal. Editing a file automatically 'bumps' the version number and a new file with the same name is created. These versions have to be explicitly 'purged' or deleted by the user, or they will multiply to fill all available space.

Directories are a 16-bit word interpreted as 2 octal numbers. Logical device names are defined in the 'SYSGEN' to be possible subsets of physical devices. Path names consist of 4 parts, the device (DR1:), the directory ([310,50]), the file name (menu.c), and a version number (;37). RSX understands default devices, current directories, and latest version numbers. So the file above, "DR1:[310,50]menu.c;37", could be called "menu.c" from DR1:[310,50] if it was the latest version. The directory SY:[0,0] is special – it is the root! It contains entries like 310050.dir that contain all the directories in the system. Access controls are primitive. Users inherit their privileges from their home directory. You are super-user if your directory is under 20. Unprivileged users can move freely about within their own directory group. ie:- [310,50] can do anything within [310,*].

Disks have 512 byte blocks. Everything is structured about them. Files can usually be seen as streams of bytes, but occasionally appear as streams of blocks.

Directory entries contain the usual pleasantries of file name and a pointer. The pointer is the number of a 'header' block at the front of the disk. The number of header blocks is set when the disk is formatted. The header says who owns the file, its access history, and where it lives. These entries are 5 bytes per extent, with overflow to another header allowed if this one fills. Extents are contiguous blocks. 1 byte is used for the number of blocks used, and 4 bytes for the relative block number of the first block. (Disks of 2 terra-bytes can be accommodated!)

The free list, like the bad block list, is implemented as a bit map.

3.3.1 Missing

— Access controls, permissions and security provisions.

— Hierarchical directories and reasonable file names.

— System wide buffering is limited to directories/bit maps.

3.3.2 Good Points

— It's quick, effective, and allows infinite file lengths.

— Applications can optimise disk access if they need to.

— Unused space is reused.

— All the disk space can be used, headers permitting.

— It lends itself to stand-alone and embedded systems.

3.4 PRIMOS

Primos does many things right. It is probably the most UNIX like of the bunch. It has a hierarchical directory structure, within MFDs (partition of a disk). Rev 19 introduced a new protection mechanism – ACLs (Access Control Lists). There are 7 permissions that can be set on each of an arbitrary list of name matching patterns. (These serve the same purpose as groups.) $REST is a catch-all. Permissions can never increase moving down a tree. 6 of the permissions are directly comparable to those of UNIX, the 7th is interesting, it allows permissions to be set. Hence it is trivial to restrict sets of users to desired environments, and still give them shared facilities like mail and print spooling. The 6 other permissions are :- read, write, execute, search, delete, and add new file. The last 3 apply only to directories and in UNIX are handled by permissions in the previous directory. Usually there is one user (the system administrator) who, like the super-user, is given unlimited access to all files. As always, system security depends on just this one id/password pair being kept secure.

The free list is stored as a bit-map called the RAT (Record Allocation Table). DSKRAT is the program used to check and fix the file system. I suspect there is more involved than just building the RAT, as it takes about 8 hours to format a 600Mb disk! This is 11 minutes for each of 44 heads, or over a second per track of 10 to 15 blocks. My guess is that all blocks have their pointers set to some known state, as well as the normal read/write cycle to check for bad blocks. Still, it is had to imagine how it takes 1 second to check out 10 to 15 blocks. Each MFD is allocated a set of heads.

Files are block based, but for text appear as a stream of bytes. Blocks contain 2k of data and a 32 byte header used for pointers.

There are a number of file types – ASCII, SAM, DAM, and ISAM. Segmented directories are a hang over from the past. They are named but are directories to unnamed, though numbered, files.

ASCII is space compressed text, but is stored as a sequential file. Many text utilities refuse to touch non-ASCII files. The editor doesn't, it will take on anything, and turn it into compressed ASCII if saved!

3.4.1 Buffering

The system does all the buffering, as in UNIX. These 'locate' buffers speed up system throughput by caching active blocks. I suspect it is write-through, not asynchronous. Only 256 buffers can be allocated. For 10 000 UFDs in 6 MFDs, the working set of buffers needed is somewhat larger than this, just for logon procedures, and caching becomes dsyfunctional.

The way directories and sequential files are stored is of particular interest here. The MFDs (Master File Directories) are special cases of UFDs (User File Directories). These are both of unlimited length, but some directory utilities, like 'ls', die when fed 1 000+ entries. A file is located by parsing the path name and first searching all the MFDs for the initial UFD. Each UFD in the path is then searched as normal. Pretty average stuff though a little clumsy at the top level. The directory entry contains ALL the info relevant to the file:- access permissions, owner, date stamps, and where to find the first block.

For a DAM file, the first block is the start of a multilevel index to all blocks in the file. For a SAM/ASCII file, each of the block headers point to the next, previous, and first blocks in the chain. They also have a length field. The first block points back to the owning directory. This intermixing of data and linkage info is a disaster in a large multi-user system. One user doing an 'ls', and getting file lengths as well, can bring the system to its knees. This is besides the locate buffers being overwritten just by reading MFDs and login files when there are many users on the system.

Power hits cause problems as well. The file system may be consistent, but the RAT will be out of date. As well, spurious end of file marks must be removed from various accounting files. As a matter of course, DSKRAT has to be run after a power fail.

3.4.2 Missing

— A single root directory.

— File links.

— Universal file display utilities.

3.4.3 Problems

— Sequential files intermix data and linkage info.

— Locate buffers are limited in scope and too few.

— Backups only write 2k blocks on tape – it takes lots of tape & time.

— SAM files only copy or backup at 1Mb / minute.

3.4.4 Good Points

— It's commercial, available, and constantly being developed.

— There is lots of software available and a lot of support.

— There are many facilities available and they are quite flexible.

— It performs well enough time sharing at moderate loads.

4. Summary

UNIX excels at what it was designed to do – multi-tasking timesharing. It's equal to the best and better than the rest at this.

In functionality, it can emulate any access method (and maybe one day V8's file system switches will be available).

In security, it can emulate most, if not all, other systems.

On raw performance, for specialised applications, it certainly loses to other systems. Contiguous tracks, giant reads and big user buffers with block structured files really do help along mammoth batch processing – if you need that sort of thing. If you want to process 200 000 transactions a day, then you either buy a batch behemoth or put in a small dedicated system that can handle 5 a second. This little system is then available for all sorts of other things.

Well that's it. If you want to timeshare, go UNIX. If you want to go batch or serious specialised database only, go elsewhere (for now).

5. Appendices

5.1 Stories

5.1.1 Hitting the brick wall

Systems which allocate files in contiguous lumps don't necessarily use all the space on disk. If there isn't /fIone/fP lump of space big enough to meet a request, regardless of the total space available, then tough luck. You've got to reorganise (or compress) the disk or whatever.

— MVS is really nice, if it can't get you what you asked for, it'll give you what it can. Which is a real bummer when your job crashes after 90 minutes CPU time because it ran out of space by a few tracks..... even though you *asked* for enough. (Oh, and of course you lose that file because the step fails and the file hasn't yet been made permanent..... Cross your fingers and submit it again!)

— On MVS a more common problem is to run out of space in a Partition DataSet. It is quite a worry the first time it happens. There you are happily editing your favourite file and you go to save it. This cryptic message 'ABEND SC037' comes back. You don't have the option to save your precious file under any other name, so you try again. Time to panic. Do you scrap the last hour's work or stay logged on and ask for help? Luckily there is a way around the problem for non-novice users of SPF (System Productivity Facility – a menu driven subsystem that almost hides TSO(Time Sharing Option) from you). Compressing a dataset (read file) is fun, if it, or the system, fails for any reason, you've lost all your data and there isn't any way to reconstruct it. Daily (full) backups are a MUST in this environment. Yes, compresses do take a long time. Not for nothing is the 'wait' symbol on 3270-type terminals a clock – set at 5 past 6. AM or PM isn't stated.

— An aside. MVS doesn't know about incremental dumps. There are a few ways to produce full dumps of disks. The most obvious is copy all files onto a tape, but this means having an up-to-date list of all the files, broken down by file type (cause you have to use different utilities to copy each file type). There is another quicker program. It produces an exact copy of the disk in its own format. These dumps are NOT transportable. They can only be reloaded onto the same device. There is no ability to reload just part of a dump – it's all or nothing here baby!

— CP/M I'm told also has its foibles. If you have a (badly) fragmented disk, then you've got to reorganise it, with all the attendant worries.

— From a brief interaction as a student, Burroughs on the B6700 also suffer this problem. It's 2pm on a Sunday, and the thing won't save your file. Time to go home and forget the assignment and passing the subject. Or maybe you could get lucky and a privileged user (read teacher) will come along and reorganise the disk for you.

5.1.2 Expiry dates

— IBM of course! MVS and DOS allow you to specify an expiry date for files on disk or tape. Running a big installation this /fIis/fP actually of great use. Tapes

will automatically re-enter the scratch pool, and disk files can't be accidentally deleted before you've consolidated the months' accounts.... Unless of course, something goes wrong with a package and processing is delayed a week, and a tape is erased but not noticed, and then the mistake is found, but by then another tape/file has slipped into the scratch pool and things become a little tricky.

— These expiry dates have 3 forms. nnn days, yy/ddd, or 99/999 for a permanent file. I wonder what is going to happen on 1/1/2000??

5.1.3 Block Sizes and Formatted Capacity

— The IBM 'key,count,data' disk format really chews up disk space. I once had a direct access file for a licensed (binary only) package that used about 100 times the amount of space it should've. It took about 150 tracks to store about 600 card images. Each track had a maximum capacity of 35k. 150 tracks stored a maximum of about 5Mb. 600 card images is about 50k. It tends to waste a lot of space.

The waste goes a little further than this. I had something like 50 card images in this file, which stored macro definitions. I figure there was about 2k characters stored on those 150 tracks. Which is an amazing low disk space utilisation. For users, it's a pity there are no data compression utilities on MVS.

— One of the big tasks on an MVS system is getting your 'blocking' factors right. You've got to figure out a whole new set of block sizes that are reasonable trade-offs between disk space utilisation and buffer size, /flevery/fP time you get new disk drives. (I've been through this exercise 6 times!) Then you have to not only move your files, but also rethink all your file design, recompile programs, and change (plus test) lots of JCL. Generally, it is lots of work.

Of course, the system programmers have to go through the same process for all the system files and libraries, and to avoid confusion (for them), change all the system file names and probably reorganise the way things are stored. And anyway there is a new operating system release to be incorporated with all its new doodads so things have to be done differently. Guaranteed, there will be quite a few surprises in store for you. Of course none of the changes will be advised to mere users, so it can take some time for all the effects to become apparent.

— The IBM supplied I/O routines impose a limit of 32k per block. This was never a problem until 2 new disk drives, the 3375 and 3380, came along with tracks larger than 32k (35k and 47k respectively). Program libraries are usually stored 1 block per track, as "it's very efficient". So all installations with these new drives had to choose between using 1 or 2 blocks per track. Each has its own advantages, both lose disk space. The best solution was to go for 2 blocks per track. The new magic numbers are 17600 and 23520.....

Seeing I'm talking about programs, it's worth mentioning that there is no space compression of uninitialised variables in stored code. If you declare a 1Mb buffer

or array, that is what is stored on disk – 1Mb of zeros..... Pretty neat if you are selling disks. It's not even worth mentioning the lack of separated program and data space, no program stack, and no possibility of shared (memory resident) libraries.

Although the code is already quite sparse, these things would make quite a difference, at the cost of totally rewriting all existing software.

An early study compared the compiled sizes of identical programs for the IBM 360 and the Burroughs B6700. The B6700 code was one fifth the size.

— Just because IBM sell a disk drive, doesn't mean that it will attach to *your* machine. And even if it does attach, it may not talk to your operating system. There is a list, of the "no warranty expressed or implied" kind, circulating that tells system programmers what will connect to what, and what all the 'magic' numbers are.

— Another tangent. MVS inherited TSO(Time Sharing Option) from OS/360. TSO is the standard interactive system for MVS. There is another (incompatible) system, called CICS(Customer Information Control System), that is good at transaction processing. A bank built it initially (it is called a 'field developed product'), so it actually performs reasonably. That its design is showing hairs doesn't matter here.

TSO is *at least* twenty years old (OS/360 was designed in 1961, and released in 1963). A recent article on TSO/E(TSO Extensions) made much of two really whiz-bang new features. It is now possible to catch the output of most commands, and to test to see if a file exists. *This* has taken twenty years to develop? Words fail me, it's just too terrible to contemplate.

5.1.4 Of Catalogs and VTOCs and things

— On MVS when a file is 'deleted' it can either be entirely removed, or just a bit removed – ie:- uncataloged and left on the pack and in the VTOC.

Usually it is quite intentional to have files not 'cluttering up' the catalogs but left about on drives. Not always. Once I had set up for me a GDG (Generation Data Group – files with version numbers) to store listings of system builds. They took 2 to 3Mb each. What to do with unwanted old versions was inadvertently specified as 'uncatalog' not 'delete'. After a few months, and a mysteriously near-filled disk, there was a rather threatening phone call insisting that all this rubbish be deleted forthwith!

— Once on a DOS/VS system, some new drives were installed. There was quite a lot of worrying going on about the size of the VTOCs, as the cylinder size was quite small. They are only allowed the last cylinder on the drive, and all your file information has to fit in there. You can run out of space on a pack by filling up the VTOC and there is no way around it. At least if all the 'inodes' are used on a

UNIX system, there are solutions. At worst the disk can be re-formatted. Preferably a few directories can be crunched with archive.

— Partition DataSets. They almost have the ability to allow access to all versions of a member. It only requires a back pointer in the directory entry and chain pointers in each old version. But that hasn't been done. You get all the overhead and problems and none of the advantages. At least on RSX and UNIVAC the previous versions of files are accessible.

5.1.5 MS-DOS

— Yes, there are quite a few good stories about this. The one I like best is the 'intelligent' default on the disk format command. Normally it prompts the user before it is about to annihilate a disk, so you really have to want to destroy data. To save the user all the effort and time of typing in the parameter "A:" then replying 'y' to the question, format charges in and re-formats the default drive WITHOUT prompting. That was probably your hard disk. And there are VERY few programs that can recover your data. My only suggestion is to take the program and hide it very well.

— Another interesting problem I encountered recently was the effects of scribbling in random parts of memory. Remember, this has no memory management. I'd forgotten to allocate space for a dynamic variable so the machine went ahead and used the area pointed to by this random variable. One time it just happened to be the main memory copy of the current directory. Boy, did it turn out to be a mess.....

— MS-DOS tends to respond to difficult situations by playing dead. It hangs with monotonous regularity. One particularly annoying habit is to insist on 'DTR' being active on the RS-232 port before doing anything else. A shame if you made a mistake and there is nothing there. So it's reboot, get a coffee, wait for all the timeouts, and eventually continue.

— Another common problem is refusing to respond to the 'reboot' key sequence. It really enters 'finger in ear' mode. I got to appreciate the RESET switch thoughtfully located on the front of the unit (NOT a standard feature). It still doesn't get over the 30 second timeout waiting for the default boot device (the floppy drive) to not respond before going and trying the hard disk. This sequence is in ROM, it's not trivial to change. The machine just sits there and waits. It's also as bad as watching it play with its memory for a minute before it finally decides to take the plunge and try to boot.

5.1.6 What can be said of RSX-11?

— Its great for many things, but etiquette forbids me using pithy phrases casting doubts on its linage or sexual practices.

I would like to publicly acknowledge its habit of keeping old versions of text files really did save me a few times. The price of administering them and their galloping desire for space is forgiven.

But aren't the directory names awful! You get used to straight jackets in time.....

5.1.7 PRIMOS

— It has lots of promise, many facilities, and a few fatal design flaws. Short of redesigning the WHOLE file system and rewriting all the utilities, there is no way of correcting the problems associated with small numbers of buffers, and a crazy way of linking sequential files.

— Not quite "file systems", but an intimate friend of it :– the editor. It really is good. It has a lot of neat tricks and is pretty quick. I've managed to produce some awfully cryptic scripts for it, to do emulate lots of utilities ranging from 'sed', 'cut', and 'paste' to generating shell scripts. Given 10 minutes, I can figure them out again. Generally top marks for a line based editor.

  It isn't a sook and will take on any file type – this is laudable until someone pokes around inside your database files and gets out the only way they know, via the save command. Bingo! squashed files. They don't read too well, but make lovely patterns on the screen. There isn't a way to undo this sort of damage – an option on edit, or a separate utility to convert between different file types would be lovely. Until then....

  But why am I complaining? It is a far, far better thing than a lot of other brain damaging software.

— Segmented Directories. There is no neat way to explain these fossilised relics of a long dead era. There are no simple utilities to work with these things. You can't tell how big in total, or in part they are. Its difficult to delete individual members, impossible to re-order them, and editing them is unthinkable.

  What are these curiousities? They are directories that live in normal UFD-style directories. They can contain named 'segments' which can be sequential files or segmented directories. A segment contains up to 64k entries identified only by their numeric position in the list. There may be gaps in the list (created by deleted entries).

  I haven't been able to get any good description of these, but can't see for the life of me why they should be used now, let alone have a major E-MAIL system (by DIALCOM) based on them.

A serial line port expander called FJ, for the Unix BSD4.2 range of workstations.

Rich Burridge

Sun Computer Australia

ABSTRACT

This paper describes the design and implementation of an RS232 port expander, called FJ, for the UNIX BSD4.2 range of workstations. Using one RS232 line on the host computer, a cable is attached to the FJ which can have upto five serial devices and one Centronics device running simultaneously.

Table of Contents

History.

One of the most popular UNIX workstations on the market today is the SUN workstation. This comes in a range of configurations. The low end machine is currently the 3/50 which has a high resolution graphics screen, a Motorola 68020 processor, 4 Megabytes of memory, SCSI disk and tape backup, Ethernet plus two RS423 ports.

This machine is capable of supporting more than just a single user but the options available for getting extra terminals connected are relatively expensive. Because this machine is bottom of the range there is no extra bus slots available to insert multiplexer boards and the like.

Extra equipment could be connected via an RS232 to Ethernet converter such as those supplied by Bridge. This is rather a costly solution. The RS423 ports on the Sun are capable of being driven at 38400 bits per second which is greater than the speed of most terminals today.

It was decided to make use of this extra speed to design a "black box" which could act as a line splitter for a variety of serial devices.

If the 3/50 could support two more ASCII terminals, a modem, a LaserWriter, and perhaps a plotter then it would be a much more complete system. It was also recognised that this would be useful on other computers which had a limited number of serial ports, and therefore was designed to work with any of the range of BSD4.2 machines.

Design.

The FJ has a single serial connection between itself and the host computer. This serial line is capable of being driven at upto 38400 bits per second but more typically at 19200 bps. It has five serial channels for peripheral devices capable of being driven from 150 bps to 19200 bps. Each of these channels is RS232-C and provide full modem control signals. There is also a Centronics port provided.

A simple packet protocol is used to send data to and from the appropriate channel on the port expander unit, to its associated process running on the host computer. The heavier the throughput on the host channel the slower the individual channels run, so even if the the terminals are set up to be running at 9600 baud, the actual output to the screen could possibly be less than this.

Each peripheral channel on the unit has internal input buffering of 1 Kbytes plus internal output buffering of 40 Kbytes. This means the unit is ideal as multiple printer spooler.

Data transmission speed on the host computer line is switch selectable. The switch is read after power-on. Changing the switch setting during operation will not affect transmission speeds.

The FJ Protocol.

All data is asynchronous 8 bit ASCII with no parity, 1 start bit and 1 stop bit.

Data packets.

Data between the host computer and the FJ unit consists of a packet with the following fields:

STX	Packet Type		Channel Number Packet Number	Count

where:

STX	Start of packet, ASCII 2.
Packet type	This is the letter 'D'.
Channel number	This is in the range ASCII 0 to ASCII 5, and takes up the top four bits of this byte.
Packet Number	This is in the range ASCII 0 to ASCII 15, and takes up the bottom four bits of this byte.
Count	This indicates how many data characters are following, and is in the range ASCII 1 to ASCII 64,thereby allowing a maximum of 64 data characters per packet.
Data	May include any ASCII character.
BCC	Block checksum. This is computed as the 8 bit exclusive OR of all the preceding characters except <STX>.

Packet Acknowledgement.

Following receipt of any type of packet except the Packet Acknowledgement packet, the response will be a packet with the following fields:

STX	Packet Type	Channel Number Packet Number	Count	BCC

where:

Packet type	This will be the letter 'A' for a positive acknowledgement, and an 'N' for a negative acknowledgement.
Count	ASCII zero. There is no data following.

Host Flow Control.

The host may send the following packet to suspend or resume data transmission.

STX	Packet Type	Channel Number Packet Number	Count	BCC

where:

Packet type	This will be the letter 'R' for X-ON, the resume command, and 'H' for X-OFF, the hold or suspend command.
Count	ASCII zero. There is no data following.

Channel Buffer Near Full.

The output buffer size available for each of the six devices is 40 Kbytes. If a buffer in the expander unit cannot absorb another block of data (within 512 bytes of full), a hold command (X-OFF) will be transmitted to the host.

After the buffer has emptied sufficiently (8 Kbytes available), the release string (X-ON) is sent.

STX	Packet Type	Channel Number Packet Number	Count	BCC

where:

Packet type	This will be the letter 'O' for X-ON, the resume command, and 'F' for X-OFF, the hold or suspend command.
Count	ASCII zero. There is no data following.

Data Rate Selection.

On power up, the data rates for each of the five serial ports are set to 9600 bits per second. This may be modified by a speed packet sent from the host.

STX	Packet Type	Channel Number Packet Number	Count	Speed	BCC

where:

Packet type	This will be the letter 'S'.
Count	This is ASCII 1. There is one character of data following.
Speed	This is in the range '0' to '7' to select one of the following speeds: 0 150 bps 1 300 bps 2 600 bps 3 1200 bps 4 2400 bps 5 4800 bps 6 9600 bps 7 19200 bps

Clear Channel Buffer.

Provision has been made for clearing out a buffer that is sending data to a peripheral. The packet for this is:

STX	Packet Type	Channel Number Packet Number	Count	BCC

where:

Packet type	This will be the letter 'C'.
Count	ASCII zero. There is no data following.

The Interface to the UNIX BSD4.2 Operating System.

On the UNIX host machine, there is a small program continually running which checks for data to be read on the serial line connected to the FJ unit. Each of the channels of the FJ is "connected" via this program, to a pseudo-tty. Data read from the RS232 line is "depacketised" and sent to the correct pseudo-tty, and any data read from the pseudo-ttys is corrected packeted and sent via the RS232 line to the expander unit.

The software uses ttyr0-ttyr6. The serial ports on the expander unit use ttyr0 - ttyr4, the Centronics port uses ttyr5 and ttyr6 is used for special commands such as clearing a channel buffer or changing the speed of an individual channel.

If a particular channel has a terminal attached, then the entry in /etc/ttys for that channel's pseudo-tty will have a login enabled. Similarily if a modem is connected to a particular channel, then an entry should be set up in /etc/remote to include that channel's pseudo-tty.

FJ Software on the Host.

fj - The Host Program.

The program on the UNIX machine that communicates with the FJ unit, is called 'fj'. This process should be started when the machine first comes up, so appropriate lines should be placed in the /etc/rc.local file.

This program has several command line switches which are:

-bbaud

this is the speed the FJ unit and the fj program communicate on the tty line. This is defaulted to 19200 bits per second.

-iinitfile

this is an alternate file containing initial speed setting for each of the FJ channels. The default is a file called .fjrc in the current directory. There is one line for each serial channel in the .fjrc file, of the form:

set speed (channel) = baudrate

For example:

set speed (0) = 9600

-lttyline

this is the name of the tty line that the FJ unit and the host fj software use to communicate. The default is /dev/ttya.

For example, if the tty line to use is /dev/ttyb, communicating at 38400 bits per second, then the fj command line will be

fj -b38400 -l/dev/ttyb

fjcontrol - A Utility Program for Special Commands.

The utility 'fjcontrol' is symbolically linked to two other programs called 'fjclear' and 'fjspeed'. These programs are provided to perform special actions on an individual FJ unit channel. They use a special pseudo-tty channel '/dev/ttyr6' to communicate with the 'fj' program, which will then set up the correct information to send to the FJ unit.

fjclear - Clear Specified Channel.

When you want to clear the internal buffer on a particular channel of the FJ unit, then you should use the 'fjclear' utility. There is one mandatory switch:

-cchannel
channel number to clear.

For example, if you have to clear the buffer associated with channel 2 on the FJ unit, then type the following command:

fjclear -c2

fjspeed - Change Speed on Specified Channel.

When you want to change the speed that a particular channel uses to communicate with its peripheral, then you should use the 'fjspeed' utility. There is one mandatory switch and one optional switch:

-cchannel
channel number to change. This parameter is mandatory.

-bbaud
this is the new speed to set this channel to. This parameter is optional and defaults to 9600 bits per second.

For example, say the want to remove a terminal from channel 1 which was running at 9600 bits per second and replace it with a modem running at 1200 bits per second, then you should type the following command:

fjspeed -c1 -b1200

fjtool - SUN Terminal Emulator Using the FJ Protocol.

There is a window based tool provided that allows you to communicate with another computer running the 'fj' host software as if it was the FJ unit. It provides two options, either a four window terminal emulation, or a two window terminal emulation with two spare channels which could be used for another purpose such as computer to computer communication. There are three optional switches:

-bbaud
this is the speed the FJ unit and the fj program communicate on the tty line. This is defaulted to 19200 bits per second.

-lttyline
this is the name of the tty line that the FJ unit and the host fj software use to communicate. The default is /dev/ttyb.

-nsubno
number of terminal emulation sub-windows to create. This can be either 2 or 4. If it is 2, then the other two channels are opened and can be used for other purposes. Default is 4 sub-windows.

For example, say we wish to connect the 'fjtool' to another Sun running the 'fj' host software. This connection is on the /dev/ttya line at 9600 bits per second. The command to type is:

fjtool -b9600 -l/dev/ttya

Future Projects.

As well as BSD4.2, a Sys V version will be completed for the FJ host software, initially using named pipes. The hardware and software will be sold not only here in Australia, but also the USA and Europe. The source of the FJ host programs will be provided with the FJ unit, because it is recognised that it can be significantly improved by UNIX experts.

A lot more can be made of the sub-window based emulators. Special versions will be written for the Apple Macintosh, the Commodore Amiga and the IBM PC.

Conclusions.

The FJ unit plus its associated software provide a relatively inexpensive solution for providing for additional ports on the Sun 3/50 workstation.

These are early days, this version of the software will probabiliy contain flaws and ineffiencies in it. If properly distributed and with contributions from experienced UNIX FJ users, this should improve in future versions.

Design of a UNIX-based Spatial Inferencing System

Gregory Toomey
Australian National University

ABSTRACT

We describe the design of a spatial inferencing system designed to run under Unix. The system is based around Sun workstations, using bitmappped screens to display geographical information. Many Unix tools have been used in the system including Remote Procedure Calls, the UNIFY database management system and Franz Lisp. We concentrate on the description of tools used and they have been combined into a prototype system.

1. Introduction

The Spatial Inferencing System is a joint project of the Department of Computer Science at the Australian National University and Centre for Spatial Information Systems at the CSIRO Division of Information Technology. The project aims to produce an expert system deducing relationships involving spatial data and display high-resolution maps of these relationships on a colour monitor. The system is multidisciplinary, involving application of computational geometry, database, graphics, and expert systems techniques in a distributed environment. Most development is carried out on Sun-3 workstations while some lisp work is being performed on a Pyramid-90x. The system consists of a number of independant modules:

Spadabas:

The heart of the system is the spatial database storing relationships between geographical entities. Spadabas, the Spatial Database, used computational geometry algorithms to retrieve the data. Data is physically stored using the UNIFY database management system.

Grint:

Grint, the Geographical Interface, can access these entities and display them as a map on the Sun-3 screen. Extensive use has been made of colours and textures to highlight geographical relationships in the map.

Reefplan

The first application to use the services of Grint and Spadabas is an expert system to assist planning land use on the Great Barrier Reef. This application is written in Franz lisp and uses Remote Procedure Calls (RPCs) to access the graphics services. A new interface has been constructed to permit lisp to converse directly with UNIFY directly via Unix pipes.

2. Spadabas - The Spatial Database.

A spatial database database describes a selection of entities contained in some global space. Eventhough we use the term *spatial* database, there is a mixture of entities that may be considered spatial and aspatial. Spatial entities have the properties of location, dimension and shape. They include roads, geographical regions, and cities. Aspatial entities do not have location ,but may be considered to be attributes of spatial objects (e.g. population density, suitability for farming). The aim was to build an integrated database incorporating spatial and aspatial entities. We used existing database technology rather than building our own database management system.

For a two-dimensional space we can classify spatial entites into three topographical types - point, line and polygon. Each type has a different set of topographical properties. The spatial entities may be partitioned into a hierarchy of subtypes, each inheriting the properties of its ancestor types. A georeference is a name for a portion of a global space. The portion is completely specified by its global properties. Different spatial entites may occupy the same physical space and so may have the same georeference. Conversely, the same space may be identified by different georeferences, possibly more than once, if the space has a number of properties in common with other spaces.

Spatial entity types have a richer set of relationships to those which have been previously treated by database technology. e.g. the proximity of one spatial object to another. These relationships can be derived from the basic topological properties of the constituents.

We first implemented SPADABAS this using the C-ISAM package. Although this worked reasonably well we later changed to the UNIFY database management system as this had greater functionality (integrated data dictionary and query language).

The Cairns section of the Great Barrier Reef has been digitised into polygonal regions and loaded into a UNIFY database. Spadabas interrogates this database using the C interface to UNIFY. One of the most common calls made by applications to Spadabas is to retrieve all the points the constitute a particular region.

3. Grint - the Geographical Interface.

Grint provides two major services:

A window manager the replaces the existing Sun window manager.

A set of high level procedure calls (running under the window manager) to draw graphical objects on the screen. Grint uses Spadabas to retrieve spatial data for displaying.

An account of Grint is given in the accompanying paper by Colin Keith.

4. Reefplan

Reefplan is a Franz Lisp based expert system designed to help planners determine land usage for the Great Barrier Reef. The system uses policy statements (rules) to zone particular regions as suitable for general use, scientific research, etc. These zonings can then be displayed on a map using the services of Grint.

The rules are stored in database along with the geographical entities. A simple facility has been contructed allowing reefplan to communicate directly with SQL (UNIFY's Structured Query Language) using pipes. A conventional SQL query such as "select * from rule/" is sent by Reefplan along a pipe to SQL. SQL will then send the answer to the query back to REEFPLAN. A filter has been placed between SQL and Reefplan to convert the SQL output into s-expressions capable of being read by the usual Franz reader. The model of this loop is:

```
Reefplan          Asks query
   |
SQL               Process Query
   |
filter       Convert query back into s-expression
   |
Reefplan          Read result of query
```

A simple timing test on a relation of 18,000 records each with 3 fields produced a retrieval rate of 41 records/second. By comparision the SQL interface to Mu-Prolog [1] retrieved 17 tuples/second on the same data.

4.1. Remote Procedure Calls

The Remote Procedure Call (RPC) facility of Sun Unix is currently being used for calls on Grint from Reefplan.

RPCs make it easy to build distributed systems. A RPC is coded just as any other procedure call. However, the processor that executes the call (the client) may be different from the processor that processes the call (the server). The parameters to the RPC must be transmitted from the client to the server, and the function return value is then transmitted back to the client. This is done using external data representation (XDR) routines which are a machine-independent way of encoding and decoding parameters. Sun have published a RPC standard used in their workstations, and a version is now distributed with for the Pyramid 90-X.

All of the functions of Grint have been coded as RPCs. Reefplan has been running on the Pyramid calling the Grint routines on a Sun-3 using RPCs. The procedures are loaded into lisp by the *cfasl* function which allows a function written in C to be called from lisp. The *cfasl* function is used to load in the RPCs into lisp.

5. A Franz Lisp Compilation Environment.

A useful feature found many lisps is a *file* facility (see [2]). Programming languages such as Pascal use a edit-compile-run cycle. In an interactive environment such as lisp, using an external editor greatly increases the time needed to correct definitions and rerun. Franz is supplied with its own structure editor to edit definitions. It is much faster and less error prone to use a structure editor than a text editor such as *vi*. However these changes are not saved when Franz is exited. Persistent objects such as functions, variables, and properties can be associated with particular source files and any changes are copied to the file when exited.

A new *Makefile* package has been implemented in Franz. It extends the existing package by supporting Liszt, the lisp compiler. The eval-when function may be inserted into the source files to specify what macros, etc. are needed for compilation.

References

1. Naish, L., *MU-Prolog 3.2 Reference Manual,* Department of Computer Science, University of Melbourne, 1985.
2. Foderaro, J.K., Sklower, K.L., and Layer, K, *The FRANZ LISP Manual,* University of California, 1983.

Development of Graphics Support for a Spatial Inferencing System

Colin H. KEITH

Department of Computer Science
Australian National University

ABSTRACT

This paper outlines the development of the Graphical Interface for the Spatial Inferencing System developed as a joint project between the CSIRO Division of Information Technology, and the Department of Computer Science in the Australian National University.

Other members of the project are:

Robin Stanton
John Smith
Hugh MacKenzie
Scott Milton
Greg Toomey
Tony Sloane

In particular, Tony Sloane and Robin Stanton have been major contributors to the graphics system.

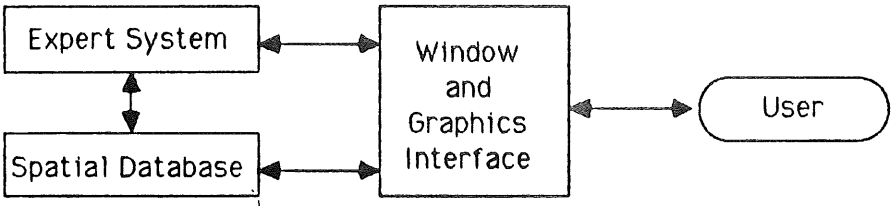
1. An Overview of the SIS Project

The system has been designed in three distinct sub-systems, the Expert System, the Spatial Database, and the Graphical Interface. Each sub-system has been implemented by different teams, using widely varying technologies. The sub-systems co-operate using agreed interfaces.

The expert system is the basic control mechanism for the entire system. It processes user queries by performing various inferencing operations and produces output by driving the graphical display. The expert system calls upon the spatial database system to perform spatial inferencing operations.

The spatial database provides general data storage and retrieval operations, and also performs a number of spatial calculation and inferencing operations. The database is accessed by the expert system for spatial inferencing and to obtain information such as corridors around objects. The graphics interface accesses the database to obtain co-ordinate data with which it draws spatial objects, and to determine the result of user selections.

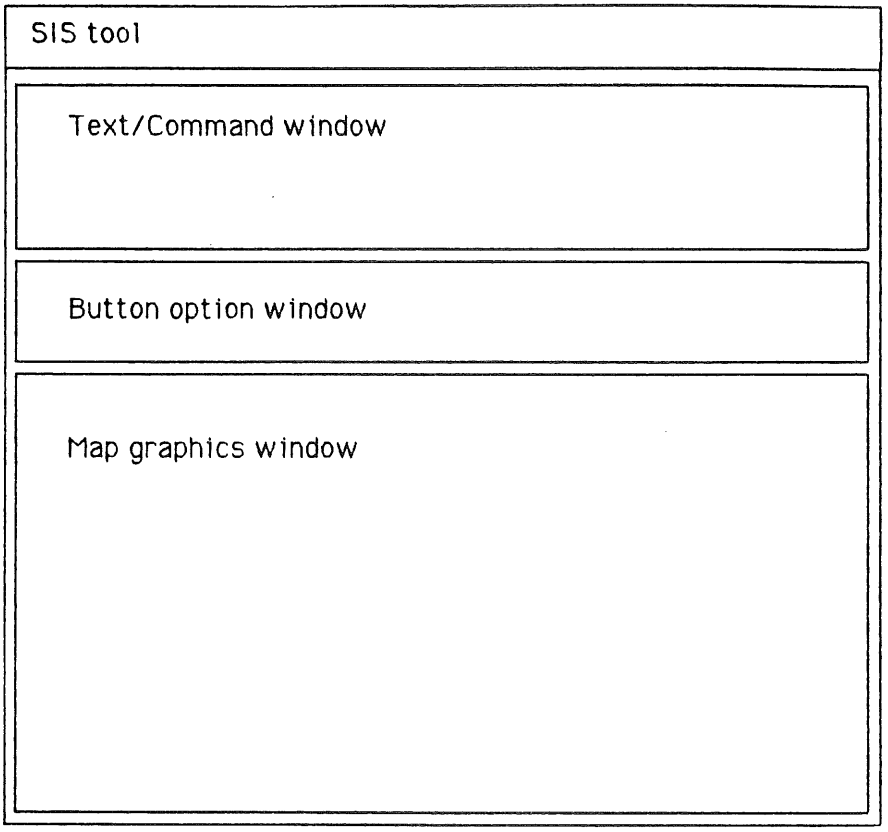
The graphical interface provides users with access to the expert system via graphical selection, in addition to textual input, and provides a medium by which users can quickly and easily see the results of their inferencing. It also allows a user to perform and control a number of inferencing experiments simultaneously. The expert system is not totally bound to the graphical system, and can be run from a non-graphical terminal.



Overview of Spatial Inferencing System

2. Project Implementation Development

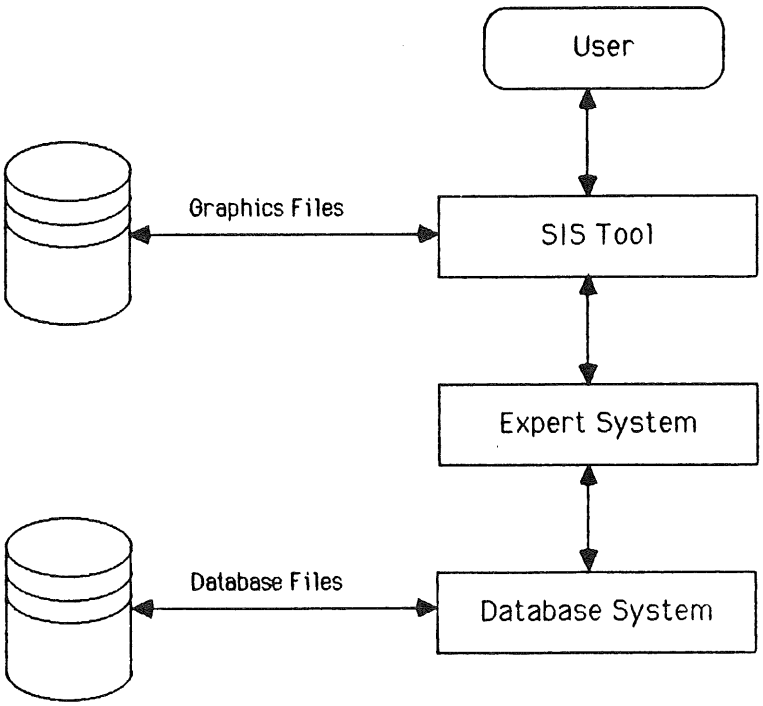
In the first instance, the system was to have a single graphical 'tool', which appeared as a single window on the display. The graphical tool was divided into three sub-windows, one to display text input from the user, and output from the expert system (the text sub-window), one to offer the user a button selection which allowed map browsing operations (the option sub-window), and one sub-window which displayed the graphical representation of the map (the graphics sub-window).



Sub-window Layout in Original Graphics Tool

The system was to be implemented on Sun-2 microcomputers, with bit-mapped graphics displays. The software to be used to implement the inital system was going to be a version of Prolog for the expert system, C code which used the Sun window software, and the Core graphics system. The software to be used for the implementation of the database system was undecided.

It was decided that the best way to approach the problem would be to build the graphical interface, with which the graphical data could be displayed and manipulated, and then to build the expert system which could use it. The database system would then be added so that the expert system could do some real inferencing on spatial relationships. (In the original plan, the graphical data would be static during user sessions, so the graphics system could use fixed data of its own, and so could operate without the database. The graphics system could then be upgraded to perform data manipulations via the database system once the database had been implemented.)



Data Flow in Original Graphics Tool Based System

2.1. Implementation of the Graphics System

In implementing the graphics system, a number of problems were encountered. Initially the major problem was a lack of experience in using the Sun windowing system, and the Core graphics system. It was also apparent that there were a number of desirable features that the system lacked.

The use of the SunCore[1] implementation of the Core graphics standard had a number of unforseen problems. When trying to run the Core graphics system in a window with other sub-windows, all of the input events (from the user) are read by the window system. The problem was that the window events also needed to be passed to SunCore. It was not possible to do this, as the input event had to be read to determine which sub-window it belonged to, and there was no way of passing the event to SunCore having already read it. (Now that there is more experience using the window system, and a source license has been obtained, it is possible for us to do this using a lower level of routines than is available to normal applications using the windowing system.)

[1] SunCore is a proprietary product of Sun Microsystems. The version of SunCore used was Sun release 1.1, revision E.

In addition to the problem of interfacing SunCore with the Sun windowing system, there were other problems particular to SunCore. Some of these were due to the actual SunCore implementation, and some were restrictions which form part of the Core graphics standard. (There was also a major problem here with a lack of experience with the standard. It was thought that the Core standard provided facilities which it doesn't.) The problems we encountered with the SunCore implementation vary from implementation restrictions (documented and undocumented) to erroneous actions.

The documented limitations (eg, maximum number of lines in a polygon, maximum number of segments) did not cause major problems in the implementation, although the limits are far too low to permit loading the data required for the initial experiments. Some undocumented limits (eg, the REAL[2] number of lines you may have in a polygon), however, caused memory violation errors, and were extremely difficult to trace. (In some cases days to weeks were lost due to the tracing and cicumvention of such problems.)

The major erroneous action in SunCore, was in the selection routine. When the user selected a line or region SunCore would, sometimes, return as the selection, an object on the other side of the screen.

The problems described here were dicovered through implementation. There were versions of the graphics tool which actually drew up sub-sets of the data, and allowed user selection, and which provided some map browsing facilities, but as a real system they were unusable because of the limitiations and faults within SunCore, and with using SunCore within a windowing environment.

2.2. Other Work

At the same time that the SunCore based graphics tool was being developed, the version of Prolog to be used for the expert system (Melbourne University's MU-Prolog) was modified to provide a means of communicating between the graphics tool and (eventually) the database system, via UNIX pipes. This modification involved learning some of the internal structure of MU-Prolog and adding primitive functions at the source code level.

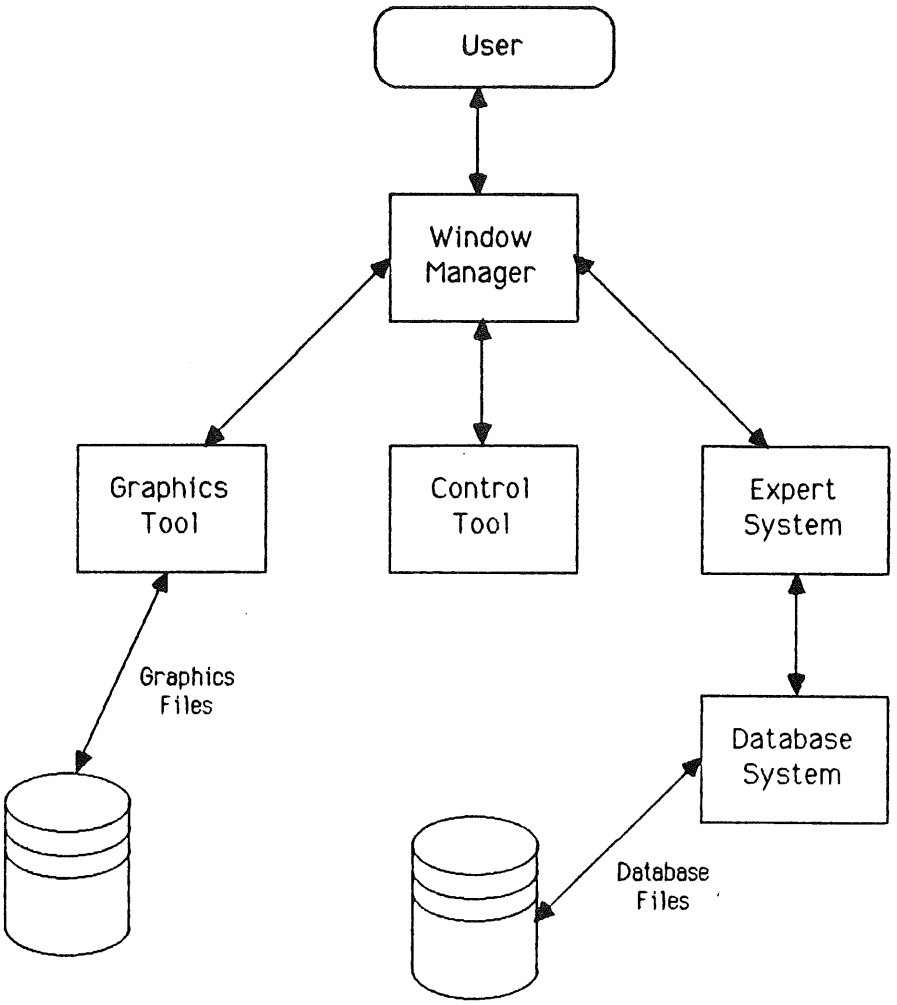
Test data was also being digitised at the time, and programs were written to convert the digitiser format to the format used by the graphics tool. In the first instance the ANU digitiser was used, and the data which was produced was not strictly consistant (ie. the lines forming a polygon did not necessarily meet where they should have). An attempt was made to produce a graphical data editor, so that the data could be massaged on the screen and made coherent. However, the data editor was using the same facilities as the graphics tool system (ie. SunCore) and was shelved as being 'too hard' until the graphics tool was operational. At this point in time there was no working database system.

2.3. The Multiple Tool Design

The decision was then made to attempt to bypass the input servicing problems in SunCore by constructing seperate tools (and hence seperate processes) for the text/command and the graphics windows. UNIX pipes were used for the inter-process communications. To set up pipes requires a single process to create a pipe, and then to fork, with each process using its one end of the shared pipe. To achieve such a system, the design was modified to be centered around an intelligent window managing process. The window manager received messages from the various processes, and rerouted them to the receiving process, so each process had to handle only one communications channel.

---

[2] The limit was expressed in documentation and error messages as MAXPOLY. When we obtained the sources, we found that this was 200, but the system would fail with a memory violation after (but not when) a polygon with more than 64 vectors was created.



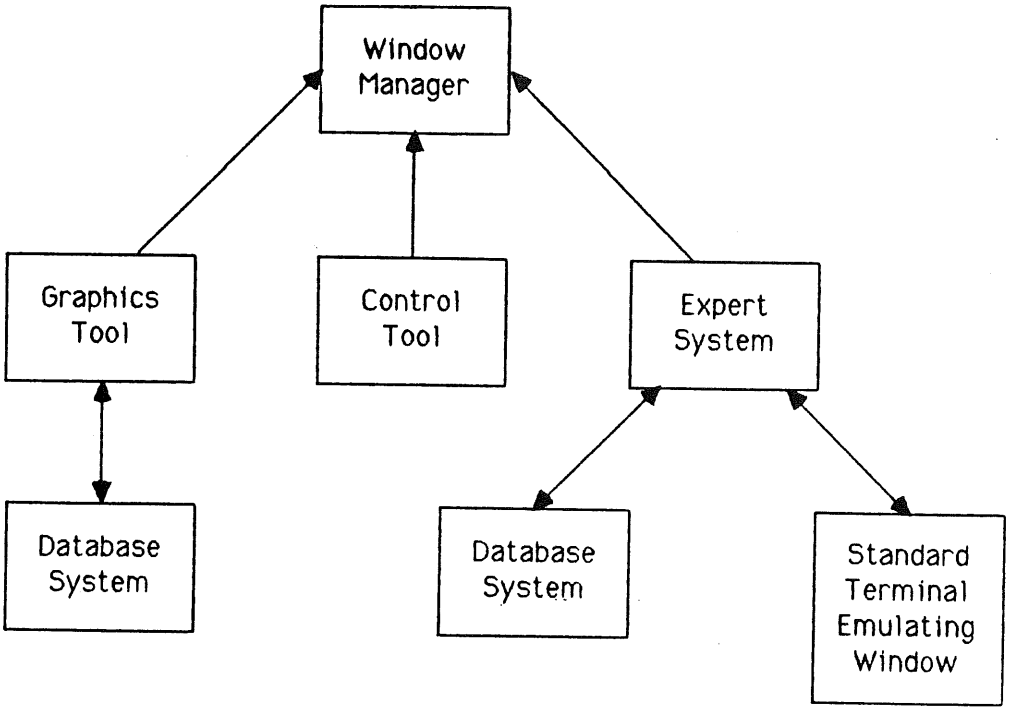
Data Flow in Window Manager Centered System

This construct had the additional advantage that it was now possible to have a number of graphics and control windows. Hence a number of expert systems could be running simultaneoulsy, with a single system being able to display different, but related, results. However, the system suffered from congestion and complexity in the window manager, as all data flow, including windowing operations, was routed through the it. The graphics tool was still designed to work from its own data files.

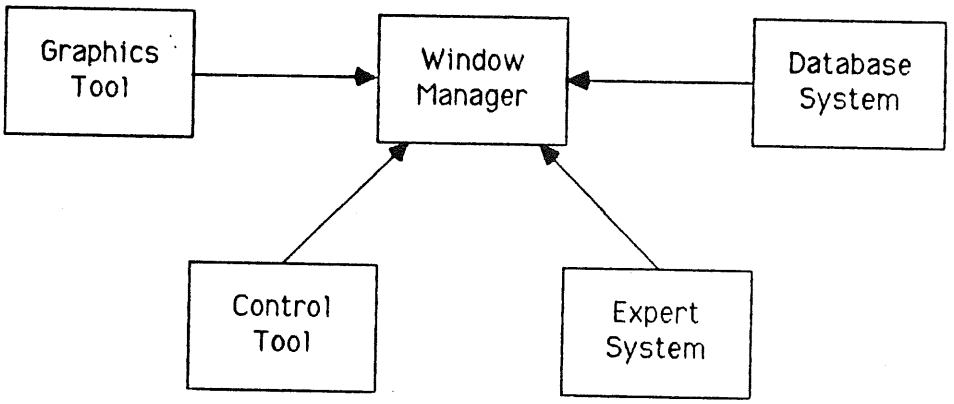
The next step was to abandon the use of SunCore a the graphics support system. The graphical interface which had been developed to load the data files into SunCore was then rewritten to perform the tasks which had previously been allocated to SunCore. This work was almost completed when the initial version of the spatial database (implemented using the CSAM system) was completed.

The utilities developed to convert the digitised data to the graphics file format were then modified to generate data files which could be loaded into the database system. The decision was then made to discard the graphics data files and for the graphics tool to obtain its data from the database. It seemed appropriate at this time to reconsider the use of the window manager as the routing control, as the amount of data which would be passed between the graphics tools and the database would be considerable, and should have a small a delay as possible.

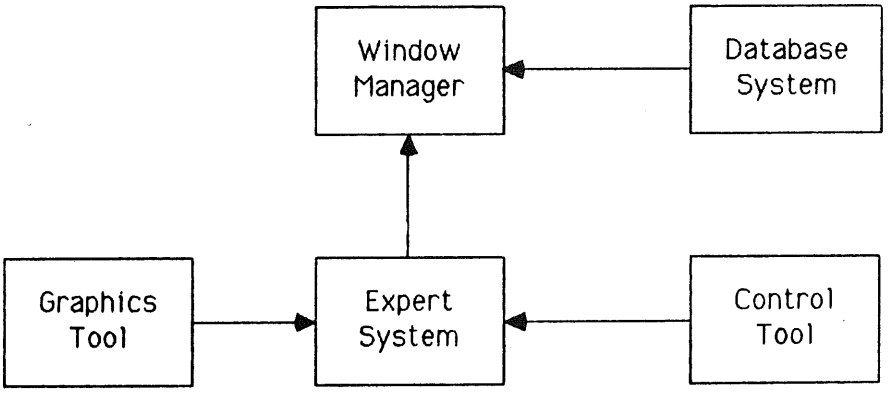
Several alternatives were considered (see "Possible Process Structures").



(a)



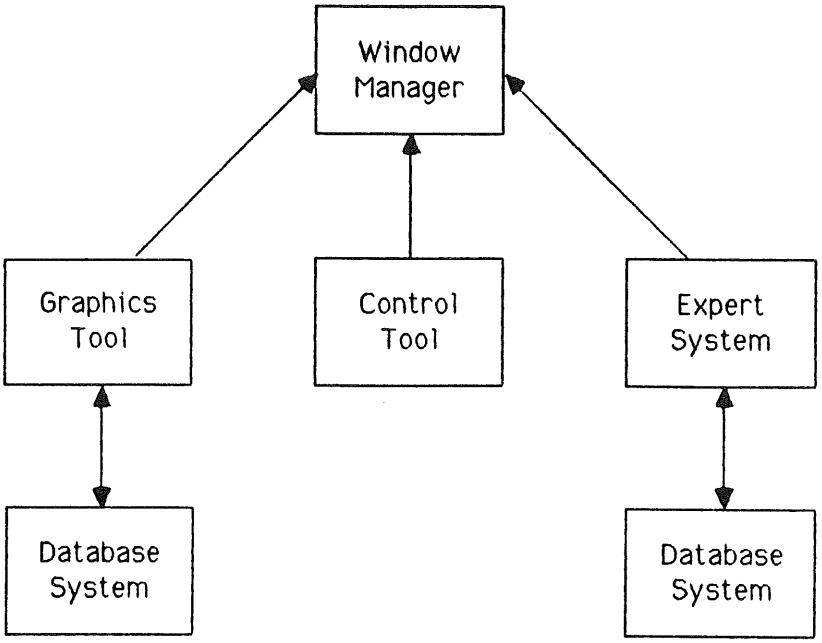
(b)



(c)

Possible Process Structures

Each alternative had the capability of having many database, graphics and expert systems. The final choice (see "Chosen Process Structure") was to maintain the structure that previously existed, and for the graphics and database applications to have their own database process.



Chosen Process Structure

The graphics tool and control tool were implemented for this configuration. The control tool passed text to the window manager, which was then interpreted as a command. The graphics tool had its own direct connection to the database, and included code to perform primitive graphics operations.

Major design decisions still had to be made with regard to the format of data to be passed between the different systems.

At the time that the data formats and protocols were being discussed, the Department gained a considerable body of knowledge about the Sun Remote Procedure Call system (The motivation being derived from two fourth year thesis in the Department). Having obtained this knowledge, it was apparent that the RPC procedural paradigm coukd be applied to the system, and that doing so would enhance the clarity of the interface between processes. It had the additional advantage that it would releive the problem of congestion int the window manager process, as procedure calls would be made directly to the serving process. A further advantage was that the RPC system is machine independant, and so it would be possible to distribute the processes across a number of machines.

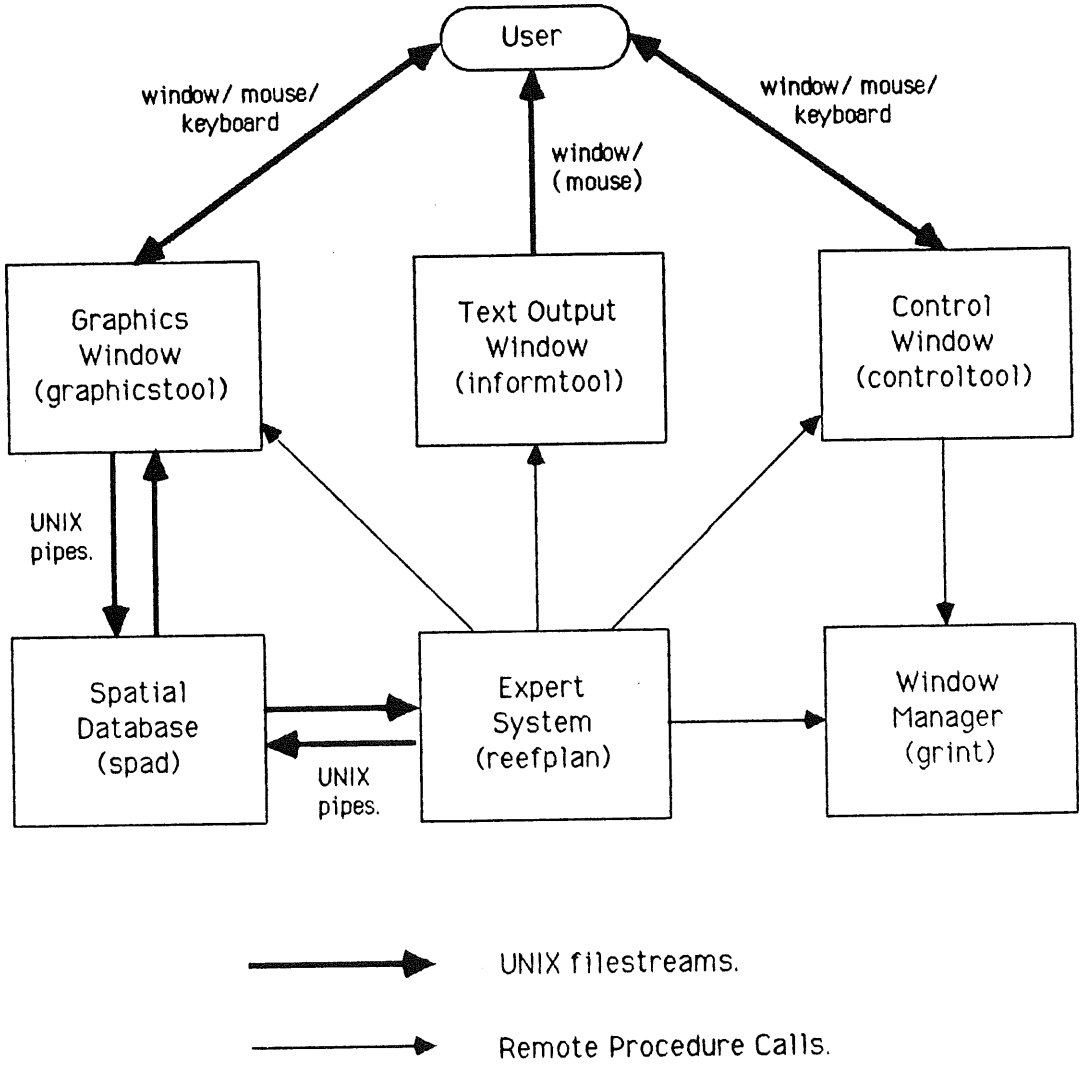
When the decision to use RPCs for inter-process communications, the pipe-based protocol for communicating between the graphics tool and the database system had already been designed and implemented. One of the major problems with implementing communications within the graphics tool was that the database access time for some operations was quite long. Consequently the implementation of the communications had to be asynchronous, otherwise user input would be blocked while database accessing was occuring.

The following chapters give more detailed explanations of the implementation of the different sub-systems and the communications between them.

3. Inter-Process Communications

As alerady described, the overall SIS system has evolved from being a single, isolated, tool (with three tightly connected processes), to being a system which operates with multiple machines, multiple processes, and multiple (but inter-related) experiments. The system currently uses both UNIX pipes and the Sun Remote Procedure Call (RPC) systems for inter-process communication.

The different mechanisms used for particular inter-process communications are given in "Current Inter-Process Communication Mechanisms".



Current Inter-Process Communication Mechanisms

Each sub-system has its own functionality and interfaces, which are described below.

3.1. The Database Sub-system

The Database System provides services to both the Expert System and the Graphics System. The Expert System makes use of the spatial inferencing functions, in addition to the storage and retrieval of objects and their attributes. The Graphics Tool component of the Graphics Sub-system uses the database for retrieving the graphical representations of objects which are to be displayed. The current implementation of communications with the database use standard UNIX pipes. A set of interface routines have been written which provide a means of serving other requests while the database access is pending. It would be possible to modify the routines to be RPC based, but the problems that this may cause (eg, blocking the graphics tool so that it will not immediately respond to user requests) have not yet been fully investigated.

The actual protocol used in the pipe is based on two-byte command and data values. The database reads the control values from its standard input, and responds with data and delimiters sent to its standard output. All data values are positive numbers in the range 1 to 32767, the control codes are positive, and the database response codes and delimiters are negative.

3.2. The Graphics Sub-system

The Graphics Sub-system is compsed of a window manager, and a number of tools which provide different services. The window manager creates all of the tools, as directed by requests which it receives via remote procedure calls or from background menu selection. It may also start an expert systems process connected to a window. The window manager provides each tool that it creates with a unique identifier by which the tool can be referred to. Each tool provides services via remote procedures calls. To hide the complexity of the remote procedure call mechanism, a library of routines is provided for each type of tool, by which the client can make requests.

3.3. The Expert Sub-system

The Expert Sub-system is the main control mechanism in the overall system. It provides no services for other processes to use (although it is hoped to provide a mechanism for asynchronous menu input), and it may use the services of all other process types.

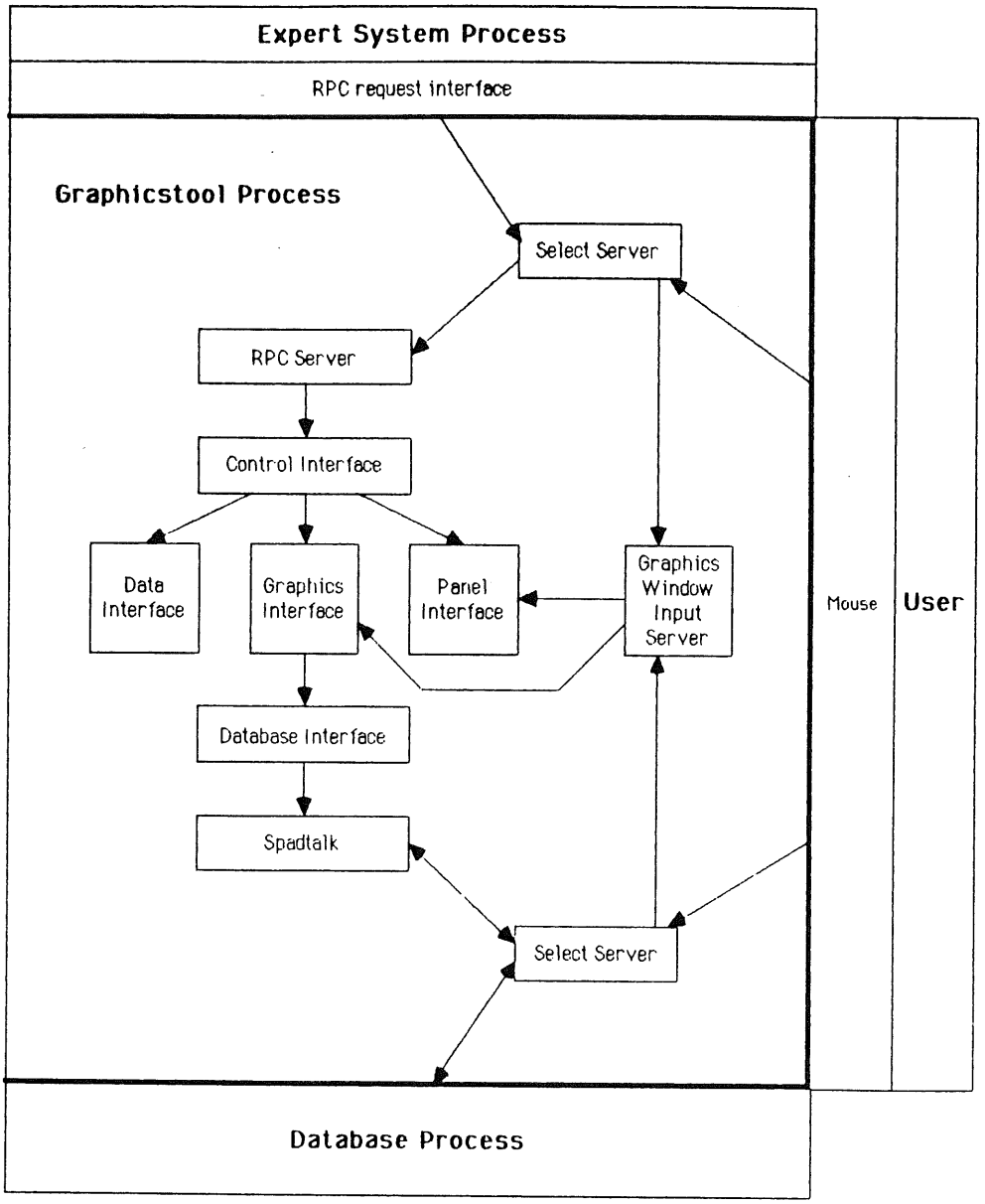
The expert system may communicate with the user via a non-graphical terminal and avoid use of the graphics system altogether. When the graphics system is in use, the user may communicate with the Expert Sub-system either through a simple terminal emulating window, or via a control window which the Expert Sub-system has set up.

4. Further Work

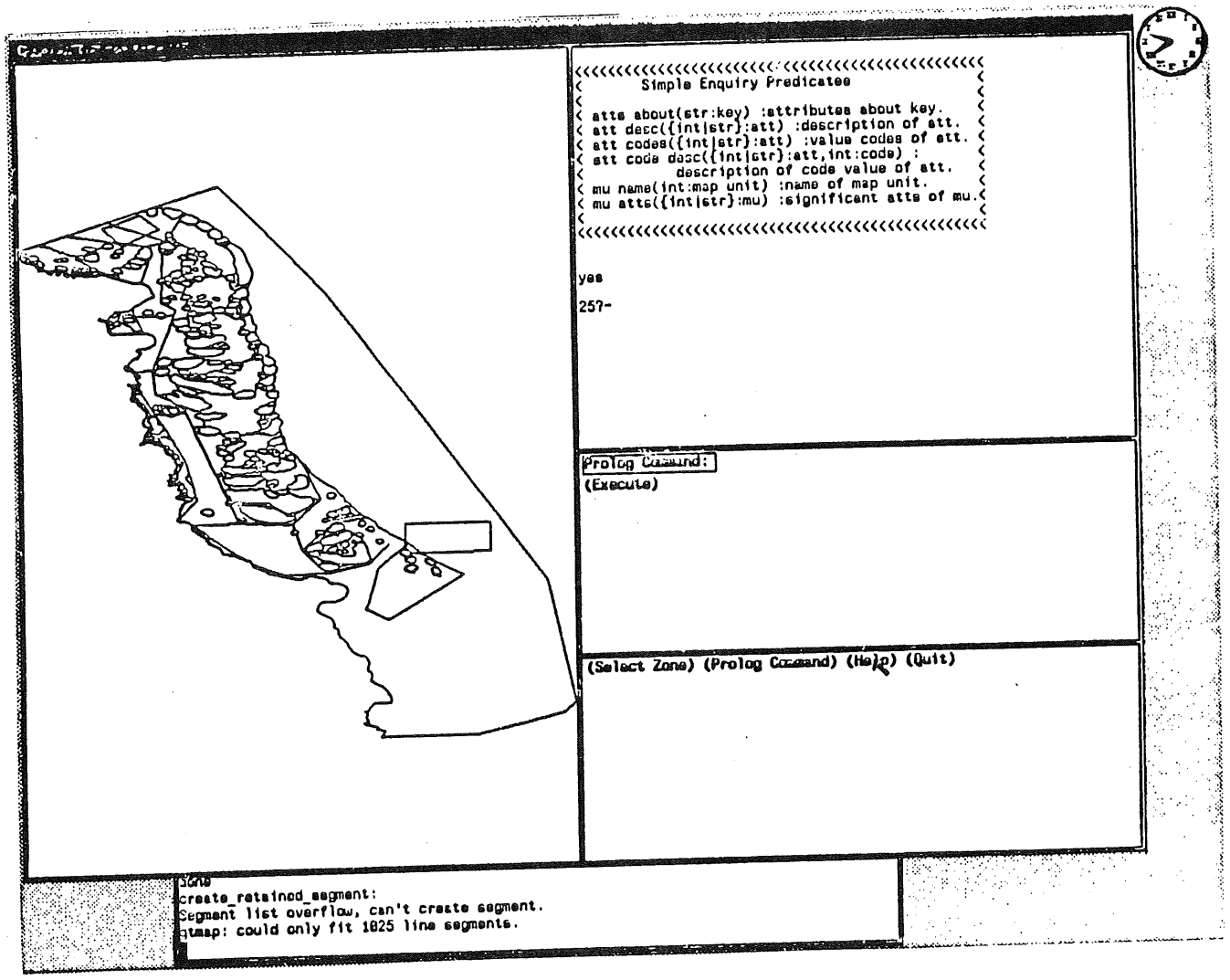
Since the original writing of this paper, the graphics tool has been extended to perform texture and colour operations. It is in the process of being rewritten to use RPC database services, and to drive several windows from the one process.

The database system has been extended to provide its services via RPCs. Hence, all of the processes can obtain database acces from a single databse server process.

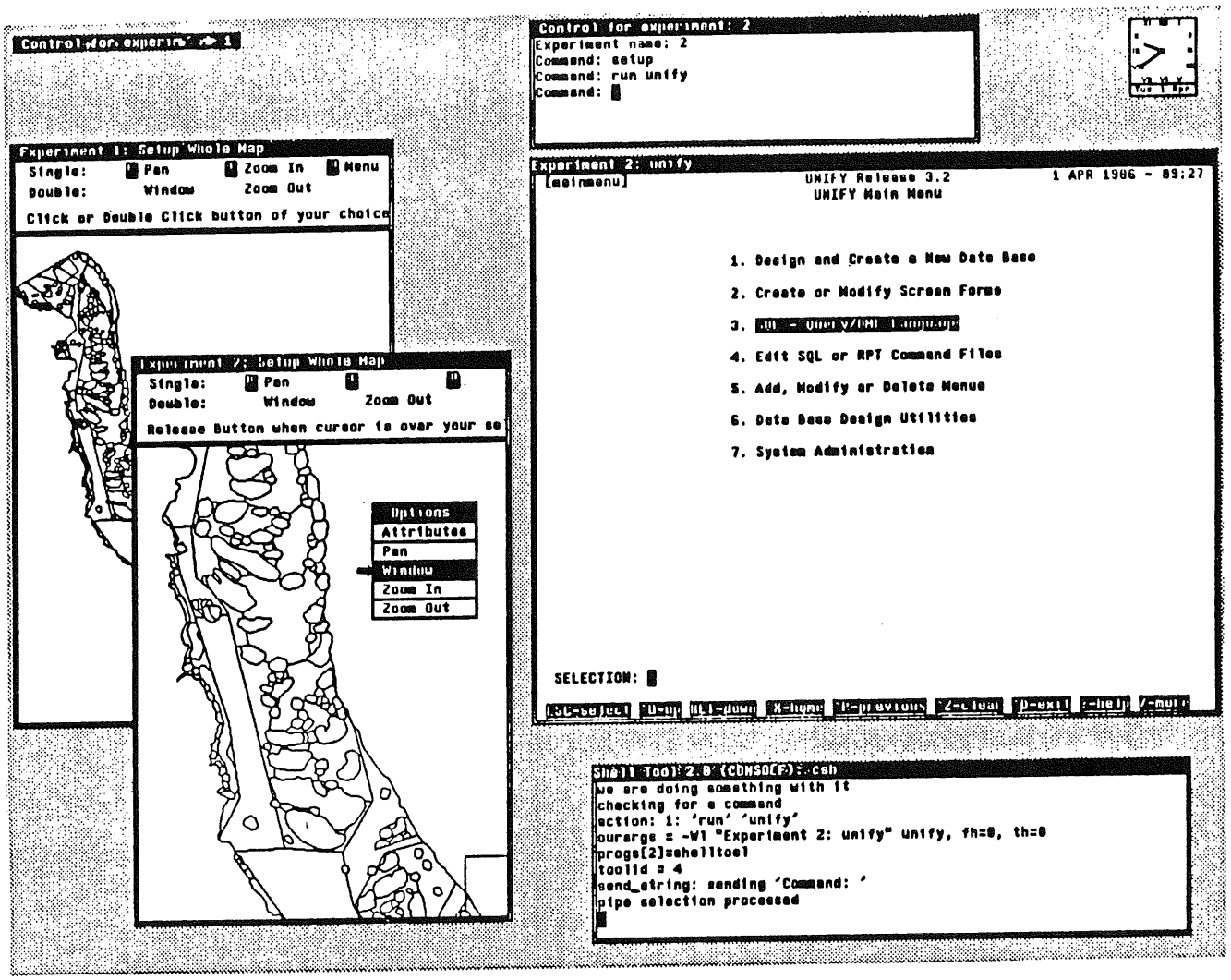
The system has been modified to use RPCs for all of its inter-process communications.



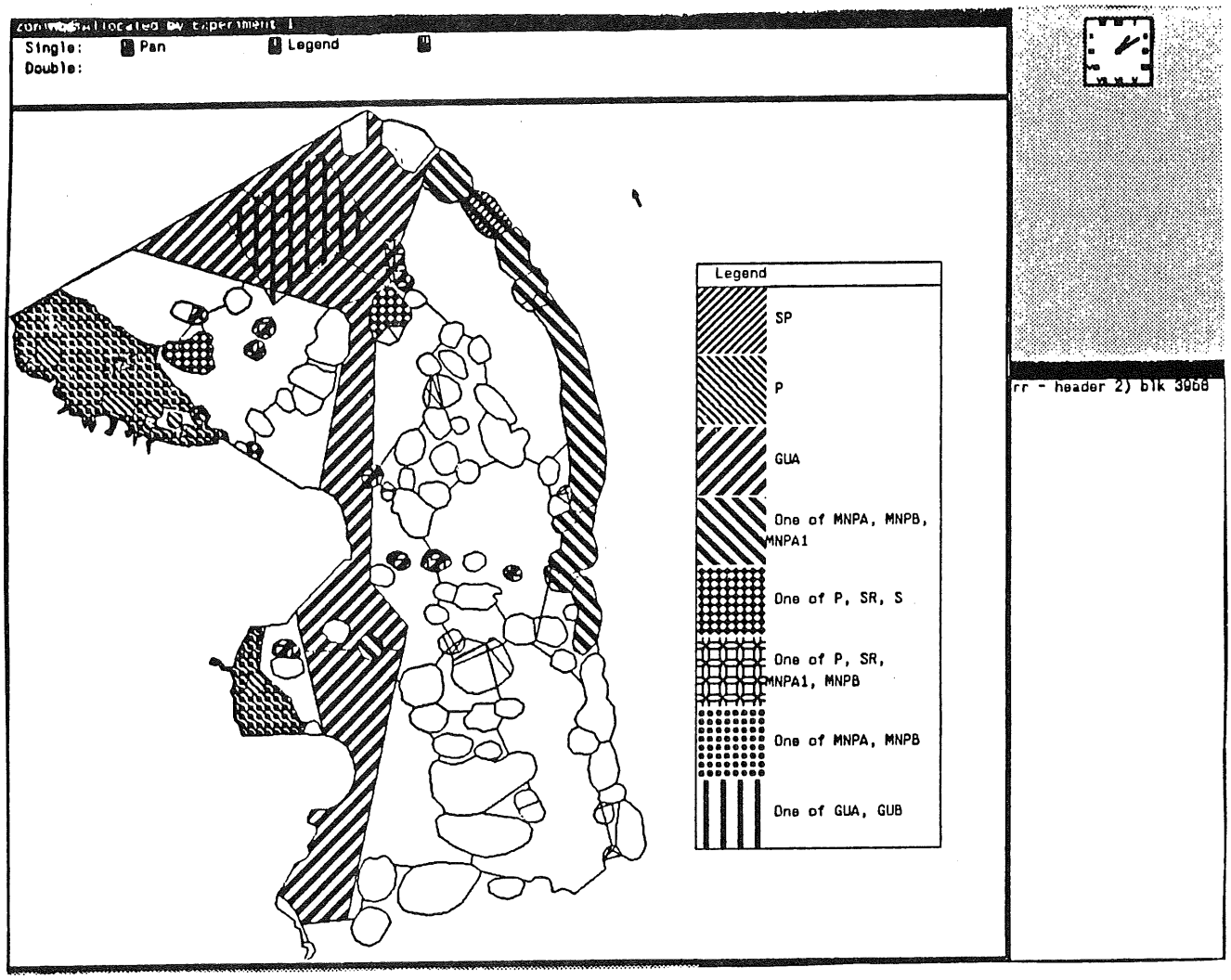
Control Flow Within the Graphics Tool Process



Snapshot of Single Process Graphics System



Snapshot of Multi-Process Graphics System



Snapshot of Multi-Process Graphics System with Current Graphics Tool

CONVEX SUPERCOMPUTER AUSTRALIA
A SUBSIDIARY OF THE LIONEL SINGER CORPORATION PTY LIMITED
Head Office: 77 Pacific Highway North Sydney NSW Telephone (02) 957 2655
PO Box 1173 North Sydney NSW 2060 Australia Telex: AA73857 FAX: 61 2 923 2570

16 September, 1986

Mr John Carey
The Editor
Unix Users Group of Australia
Monash University
Wellington Road
CLAYTON VIC 3168.

Dear John

I recently received from the United States, an excellent paper prepared by two technical programming experts with Convex Computer Corporation, Richardson, Texas.

I have enclosed a copy of this Abstract as I'm sure it will create considerable reader interest when you are able to publish it.

Yours sincerely,

LIONEL SINGER
Managing Director.

Encl:

LS:ND

Porting the 4.2BSD UNIX Virtual Memory Subsystem

James E. Mankovich
Robert B. Kolstad
Convex Computer Corporation
701 Plano Road
Richardson, Texas 75081
(214)952-0200

ABSTRACT

UNIX is a general-purpose, multi-user, interactive operating system that is rapidly becoming the industry's standard. Because of these and other characteristics of UNIX, it was chosen as the operating system to be used by CONVEX Computer Corporation and its customers.

This document describes the work entailed in porting the VM subsystem of 4.2 BSD UNIX from the VAX architecture to the CONVEX architecture. It explains the assumptions uncovered about the VAX VM architecture within the 4.2 UNIX kernel as well as how these assumptions were resolved in its port.

Goals

CONVEX Computer Corporation required an operating system for its new affordable supercomputer. The choice of designing a new operating system was an unpleasant alternative to porting an existing one. Berkeley UNIX, with its virtual memory and other enhancements, became the system of choice for CONVEX. The operating system group committed to an ambitious implementation schedule which required the operating system to be up and running two months after the machine passed diagnostics.

Aside from the overall goal of porting the Berkeley VM subsystem to the CONVEX architecture, several intermediate requirements were established:

1 Change as little of 4.2BSD UNIX as possible
2 Adhere to the high level paging/swapping algorithms.
3 Isolate and remove machine dependencies from the VM Subsystem
4 Support 128 MB of physical memory and up to 2 GB processes
5 Utilize CONVEX hardware 'referenced' bits to enhance paging performance

Possible future Berkeley UNIX releases motivated the decision to change as little of 4.2BSD UNIX since CONVEX may want to upgrade in the future. By modifying only what is required in order to perform the port, it was thought to be easier to incorporate any bug fixes or enhancements made to UNIX in the future.

Since the 4.2BSD UNIX VM Subsystem has been field tested by many sites, we anticipated higher reliability and performance by adhering to its high level paging and swapping algorithms. The isolation and removal of machine dependencies from the VM subsystem was set as a goal because of the anticipated need to port UNIX to yet another virtual memory architecture in the future.

The support for large physical memories and virtual process sizes utilizes the capabilities of the CONVEX C-1 architecture. By utilizing hardware "reference" bits (as opposed to simulating them as is done in 4.2BSD VAX UNIX), it was hoped that system paging performance would be improved.

CONVEX vs. VAX VM Architecture

Both the VAX and the CONVEX computers have virtual memory architectures with a 32 bit virtual address space. Each implements this space using page tables to perform virtual

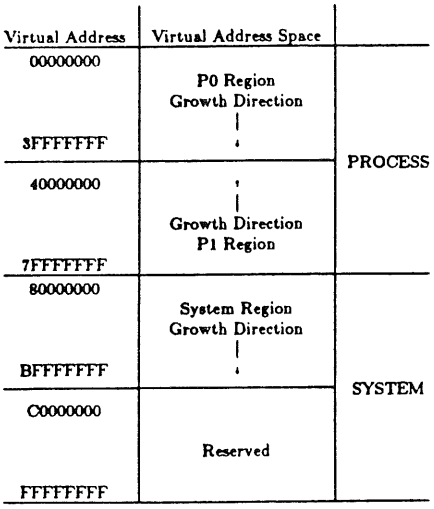
address translations. The list below outlines the differences between the physical and virtual memory block size, virtual address translation mechanism, and user/system protection mechanisms:

- o VAX has 0.5 KB vs. CONVEX 4 KB pages.
- o VAX virtual memory uses contiguous page tables and segment length registers. CONVEX uses double level page tables with integral "valid" bits.
- o The VAX uses hierarchical processor modes for virtual memory protection; CONVEX uses hierarchical rings.
- o The VAX has hardware modified bits and no referenced bits; CONVEX has both hardware modified and referenced bits.

The total virtual address space on both the VAX and the CONVEX is 4 GB. The two virtual memory architectures differ in the way in which the 4 GB address space is partitioned between process and system as well as in the mechanisms used for virtual address translation.

The VAX global virtual address space comprises two 2 GB address spaces denoted *process* and *system*. The *process* address space further splits into two 1 GB regions know as the P0 and P1 regions. These three regions (P0, P1, and *system*) define the total extent of software accessible virtual memory on the VAX. Figure 1 illustrates the layout of the VAX global virtual address space.

Figure 1: VAX Virtual Address Space

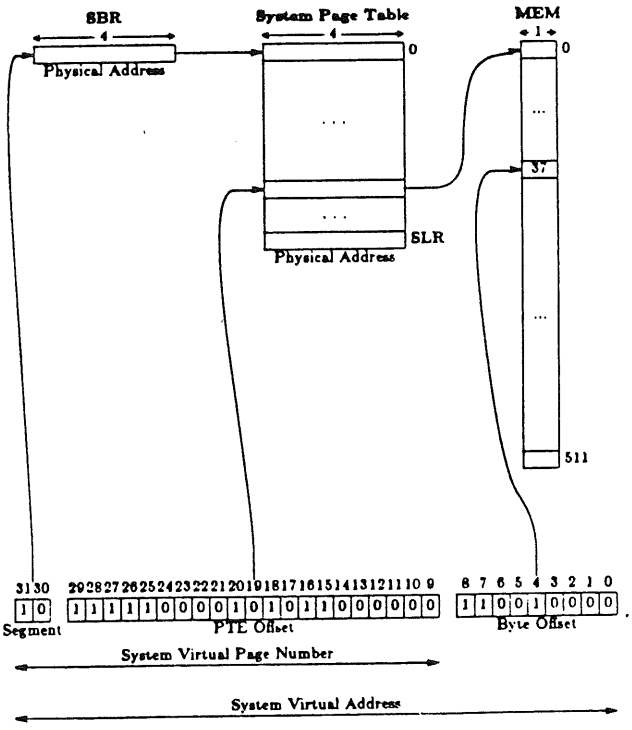


VAX System Address Translation

The system page table (SPT) which maps the *system* virtual address space is a contiguous array of page table entries (PTEs). The system base register (SBR) contains the physical address of the system page table. The system length register (SLR) contains the number of PTEs within the system page table. Each PTE within the system page table maps 512 bytes of physical memory, so the total extent of system virtual memory is 512*c(SLR) bytes.

Figure 2 illustrates system virtual address translation. The first 23 bits (2 for segment, 21 for PTE offset) of the virtual address determine the System Virtual Page Number (SVPN). The two-bit segment portion of the SVPN chooses the SBR which contains the physical address of the base of the system page table. The PTE offset portion of the SVPN chooses a PTE from the contiguous array of PTEs stored at the base address. This physical address within the selected PTE forms the all but the bottom 9 bits of the physical address of the data entity desired; the bottom 9 bits are the same ones as the byte offset in the system virtual address.

Fig. 2: VAX System Virtual Address Translation

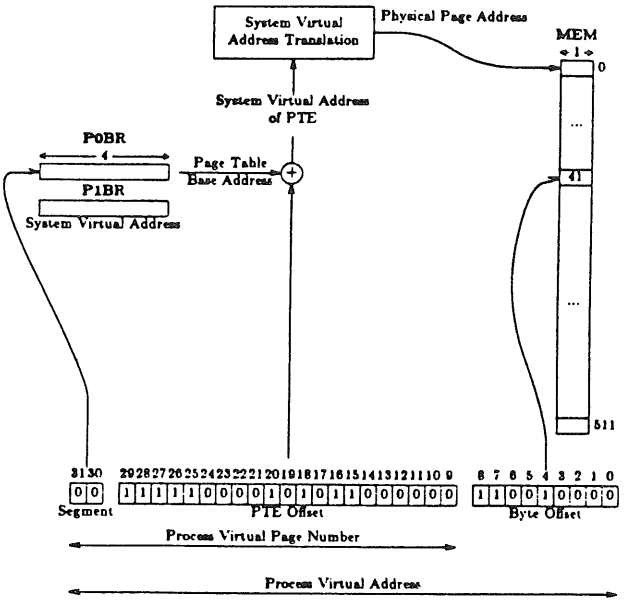


VAX Process Address Translation

Two independent page tables, POPT and P1PT map the *process* virtual address spaces called P0 and P1, respectively. The base registers P0BR and P1BR contain the *system* virtual address of the base of the *process* page tables (n.b. the difference between system virtual address, process virtual addresses, and physical addresses throughout this discussion). The length registers P0LR and P1LR define the extent of the process page tables. The P0 and P1 regions grow toward each other, the P0 region being used for data and the P1 region for stack.

Since the process page tables exist within the system virtual address space, process virtual address translation is a two level address translation scheme using both system virtual addresses and process virtual addresses. Figure 3 outlines the steps performed in translating a Process Virtual Address. The first step generates the *system* virtual address of the *process* PTE associated with the *process* virtual address. The second step generates the physical address of the addressed entity using the contents of the PTE loaded from *system* virtual memory. This mechanism permits each process page table page (512 bytes) in system virtual memory to map 64 KB of process virtual memory.

Figure 3: VAX Process Virtual Address Translation



The length registers SLR, P0LR, and P1LR control the validity of any virtual address within a region by defining the length of the page tables. Each page table entry specifies the residency and access privileges of the associated virtual page.

CONVEX Virtual Memory

The CONVEX virtual address space comprises eight 512 MB segments numbered 0 through 7. Of these 8 segments, 4 define the process address space and 4 define the system address space (though the address interpretation is uniform). Table 4 shows the virtual address space layout of the CONVEX architecture.

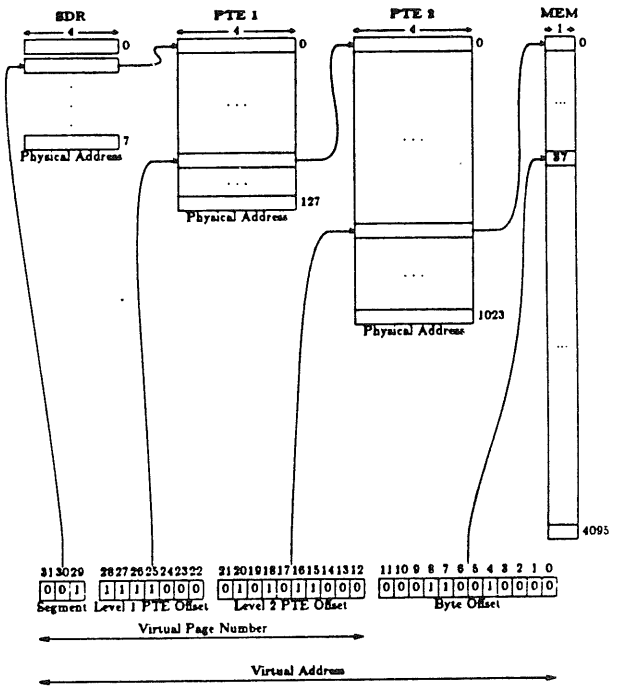
Table 4: CONVEX Virtual Memory Space

Virtual Address	Virtual Address Space	
00000000 1FFFFFFF	Segment 0	SYSTEM
20000000 3FFFFFFF	Segment 1	
40000000 5FFFFFFF	Segment 2	
60000000 7FFFFFFF	Segment 3	
80000000 9FFFFFFF	Segment 4	PROCESS
A0000000 BFFFFFFF	Segment 5	
C0000000 DFFFFFFF	Segment 6	
E0000000 FFFFFFFF	Segment 7	

A set of 8 Segment Descriptor Registers (SDRs) map the virtual address space. Each SDR contains the physical address of 128 contiguous PTEs which compose the first level page table for the segment. Each first level PTE points to a contiguous array of 1024 PTEs which is the second level page table. The SDRs control the validity of each 512 MB Segment; the first level PTEs control the validity of 4 MB regions within each segment; and the second level PTEs control the validity of each 4 KB page within the 4 MB regions.

The second level PTEs associated with the virtual page control the validity, residency, and access privileges of a given virtual address within a valid segment. First level PTEs control the validity and residency of 4 MB regions; second level PTEs control validity, residency and access privileges of 4KB pages. Figure 5 depicts the CONVEX virtual address translation mechanism for all virtual addresses.

Figure 5: CONVEX Virtual Address Translation



Virtual Memory Protection Mechanisms

Virtual memory protection is the function of validating whether a particular type of memory reference is to be permitted to a particular page. The VAX and CONVEX architectures support two types of memory protection:

- o Protection from invalid memory references
- o Memory access privilege verification for valid references

Protection from invalid memory references is a requirement for a robust operating system environment in which the system is protected from process address references. Access protection provide a means for enforcing specific reference characteristics of particular virtual memory pages, e.g., read, write, or execute access.

The VAX architecture uses hierarchically ordered processor modes coupled with PTE protection codes to provide virtual memory protection. The processor can be in one of the following mutually exclusive four modes: kernel, executive, supervisory, or user. kernel mode is the most privileged mode and user the least privileged.

Associated with each virtual page is a PTE protection code which describes the accessibility of the page for any given processor mode. The protections codes permit memory access protection within the following limits:

- o Each mode's access can be read/write, read only, or no access
- o If any mode has read/write access, then more privileged modes have read/write access
- o Read access implies execute access

Figure 6 describes the available PTE protection codes implemented by the VAX architecture.

Figure 6: VAX PTE Protection Codes

MNEMONIC	K	E	S	U
KW	RW	–	–	–
EW	RW	RW	–	–
SW	RW	RW	RW	–
UW	RW	RW	RW	RW
KR	R	–	–	–
ER	R	R	–	–
SR	R	R	R	–
UR	R	R	R	R
ERKW	RW	R	–	–
SRKW	RW	R	R	–
URKW	RW	R	R	R
SREW	RW	RW	R	–
UREW	RW	RW	R	R
URSW	RW	RW	RW	R

– = no access
R = Read Only
RW= Read/Write

K = Kernel
E = Executive
S = Supervisory
U = User

The Berkeley UNIX implementation on the VAX utilizes only 2 of the 4 possible processor modes: kernel and user. This reduces the possible PTE protection codes to the set shown in Figure 7.

Figure 7: VAX PTE Protection Codes used by UNIX

MNEMONIC	K	U
KW	RW	-
UW	RW	RW
KR	R	-
UR	R	R
URKW	RW	R

- = no access
R = Read Only
RW= Read/Write
K = Kernel
U = User

The VAX has a set of machine instructions for changing "processor modes". These instructions permit changes from one processor mode to an equivalent or more privileged processor mode. UNIX uses the "change mode to kernel" instruction (chmk) to enter the kernel from user space in a controlled fashion. The return from interrupt instruction (rei) causes the processor to revert to an equivalent or less privileged processor mode. UNIX uses this instruction to return to user mode after a system call or context switch. These two instructions (chmk and rei) provide the basis for the UNIX user/kernel protection mechanism as well as the framework for the system call and scheduler implementations.

CONVEX Protection Mechanisms

The CONVEX architecture uses a hierarchical ring structure along with a set of addressing rules based on these rings to provide protection between the kernel and user. The eight segments are divided into five rings as shown in figure 8.

A set of addressing rules know as ring maximization controls the validity of a memory reference made while the processor is executing within a specific ring. The concept of ring maximization stipulates that the current ring of execution defines the extent of memory access privileges. The hierarchical ordering of the rings dictates that higher priority rings have access to all rings of equivalent or lower priority. Figure 9 shows the validity of memory references given the current ring of execution.

Figure 8: CONVEX Ring Structure

	Virtual Address	Virtual Address Space	
Ring 0	00000000 1FFFFFFF	Segment 0	SYSTEM
Ring 1	20000000 3FFFFFFF	Segment 1	
Ring 2	40000000 5FFFFFFF	Segment 2	
Ring 3	60000000 7FFFFFFF	Segment 3	
Ring 4	80000000 9FFFFFFF	Segment 4	PROCESS
	A0000000 BFFFFFFF	Segment 5	
	C0000000 DFFFFFFF	Segment 6	
	E0000000 FFFFFFFF	Segment 7	

Figure 9: CONVEX Ring Maximization

Ring of Execution	Memory Reference Validity				
	Ring 0	Ring 1	Ring 2	Ring 3	Ring 4
0	V	V	V	V	V
1	-	V	V	V	V
2	-	-	V	V	V
3	-	-	-	V	V
4	-	-	-	-	V

- = Invalid Access V = Valid Access

In relation to the VAX architecture, the rings can be thought of as processor modes with ring 0 being the most privileged mode (kernel) and ring 4 being the least privileged mode (user). The major difference between the CONVEX and VAX architecture is that the PTE protection bits on the CONVEX are not mode dependent. Only memory references which are considered valid by ring maximization proceed to use the PTEs to determine access privileges.

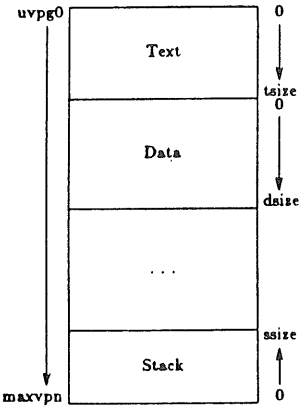
Since the current ring of execution determines access privileges, a ring crossing mechanism exists to protect against illicit ring crossings. A ring crossing can only occur by executing an explicit instruction which crosses rings or by an

exception which requires kernel services. The CONVEX architecture supports a cross-ring procedure call/return mechanism which permits only inward calls (towards ring 0) and outward return (toward ring 4). By using the CONVEX cross-ring call/return mechanim and placing the UNIX kernel in ring 0 and user processes in ring 4, the CONVEX protection mechanism directly maps the VAX protection mechanism.

Architectural Assumptions within Berkeley 4.2 BSD UNIX

The UNIX VM subsystem partitions a process's address space into three independent segments: text, data, and stack. These segments have respective extents of tsize, dsize, and ssize. See Figure 10 for a visual depiction of these segments. There are two schemes for defining a page's number. One scheme indexes the pages linearly starting with 'uvpg0' and continuing through the final stack page 'maxvpn'. This number is the "process virtual page number". The other scheme numbers the pages within a segment (forwards for text and data, backwards for stack). Each process has a virtual page zero (on the VAX this is 0) where the text starts and a maximum virtual page number (on the VAX this is 0x7FFFFFFF) where the stack starts (the stack grows toward lower memory). Within 4.2 BSD UNIX there is a set of macros (in *vmmac.h*) for translating between virtual page numbers and segment numbers and vice versa.

Figure 10: 4.2 BSD Process Layout



The initial step in porting the 4.2 BSD UNIX VM subsystem was the isolation of the architectural dependencies. One basic premise of the VM subsystem was that a process virtual page number could be generated given a specific process id and system virtual PTE address. The VAX architecture satisfies this assumption because the process page tables are contiguous within system virtual memory. Subtracting the system virtual PTE address from the base of the process page table in system virtual memory generates the zero relative process virtual page number. Virtually all the VM routines within VAX UNIX pass a PTE address and count as parameters. If a subroutine needs a process virtual page number, the PTE address and process id easily generate it.

The CONVEX double level page table address translation scheme does not easily permit virtual page number generation from PTE addresses because the second level page tables can reside anywhere within physical memory. Without forcing all process page table pages to be contiguous on the CONVEX architecture, it is very difficult to generate the virtual page associated with the PTE from given a PTE address and process id. Since non-contiguous process page tables were desired (i.e., page table pages were to be allocated the same way as general memory pages), CONVEX UNIX could have no assumptions about contiguous page tables.

The next step (and most difficult part) of the VM port was the removal of the assumption that a virtual page number could be generated from a PTE address. Removing this assumption required two changes. The first was the removal of a set of macros in *vmmac.h* for converting a PTE to a virtual page number. The second was to change the VM subroutines which used these macros to accept the virtual page number as an argument. By passing only virtual page number between the VM subroutines, those routines which required PTE addresses could easily generate them. After removing the PTE to virtual page translation assumption, it became much easier to port the VM system to other architectures since there is always a way to generate a PTE address given a virtual address.

The use of VAX modified bits constituted another machine dependency. On the VAX architecture, the PTE which points to a physical page contains that page's modified bits. VAX UNIX deals with the modified bits on a virtual page basis since they are associated with a PTE on the VAX. The CONVEX modified bits are not contained within the PTE's but rather are located within the memory mapped I/O portion of memory. In order to deal with the modified bits independently of virtual pages, CONVEX UNIX uses five new macros to perform all modified bit

manipulation. The five macros are:

- o modified - Returns the state of the modified bit
- o setmodified - Set the state of the physical page to modified
- o clearmodified - Set the state of the physical page as unmodified
- o anyclmodified - Is any page within a cluster modified
- o distclmodified - Distribute modified bits to all pages in a cluster

These macros remove the architectural dependencies of VAX UNIX. Their correct definition allows the VM code to be used on any VM system.

Implementation

Because hardware for the CONVEX computer was not to be available for several months, CONVEX augmented the instruction set simulator constructed for validating compilers. Adding a complete simulation of the CONVEX virtual memory system required less than two working days. Simulating the operating system required about 20 minutes of CPU time to reach the first prompt (from the shell spawned by init). The first system message contained an interesting twist on the nUxi problem: "Unix Version 1" printed as "xinUreV nois 1" due to the 32 bit word transpositions between the simulator and the simulated I/O system.

The virtual memory project required about five man-months to complete. Most routines were unchanged; only two required major modification. Virtual memory exercising test suites aided debugging in final implementation. The system is now in production use and has shown no discernible failures on 75 machines over periods ranging up to two years.

Vector Register Scheduling

Since the CONVEX architecture provides integrated vector processing, the vector registers (8 registers, each 128 elements by 64 bits) have to be shared among all the processes. In order to reduce the overhead incurred on saving the vector registers in memory when context switching, the C-1 hardware provides a system trap for vector register access.

The vector register access trap is controlled by two instructions, one to enable or disable a vector valid trap, and one to test the current state of the vector valid trap. When vector traps are enabled, any access to the vector registers cause the kernel to be entered. When vector traps are disabled, access to the vector registers proceeds normally.

The CONVEX UNIX kernel uses the vector valid trap mechanism to defer the saving and restoring of the vector registers until absolutely necessary. Also if a process does not use of the vector registers, no vector registers saving is done.

When a process is scheduled for the first time, it is started with vector valid traps enabled. Only when a process takes a vector valid trap is the process considered a vector program, and vector register scheduling required. If a vector valid trap occurs for process A and process B is currently holding valid vector state in the vector registers, the vector registers are flushed to process B's u area and then allocated to process A which trapped. Process A is then scheduled immediately with vector traps disabled so it may use the vector register.

Conclusion

The CONVEX UNIX kernel uses a new style of virtual memory addressing with double indirect page tables which allows the second level page tables and the data pages themselves to be paged out. A modified system with generalized virtual memory macros is arguably more "portable". This kernel repairs several virtual memory latent bugs and has now been run for years in a production environment.

THIS PAGE INTENTIONALLY LEFT BLANK

AUUG

Application for Ordinary, or Student, Membership Australian UNIX* systems Users' Group.

*UNIX is a trademark of AT&T Bell Laboratories.

To apply for membership of the AUUG, complete this form, and return it with payment in Australian Dollars to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

Please don't send purchase orders — perhaps your purchasing department will consider this form an invoice. Foreign applicants please send a bank draft drawn on an Australian bank, and remember to select either surface or air mail.

I, ... do hereby apply for

- ☐ Renewal (indicate which membership type).
- ☐ Membership of the AUUG $ 50.00
- ☐ Student Membership of the AUUG $ 30.00 (note certification on other side)
- ☐ International Surface Mail $ 10.00
- ☐ International Air Mail $ 50.00

Total remitted AUD$_____
(cheque, money order)

I agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

I understand that membership includes a subscription to the AUUG newsletter, and that I will be entitled to attend AUUG sponsored functions at member rates for the duration of my membership.

Date: ___/___/___ Signed: _____

☐ Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

For our mailing database - please type or print clearly:

Name: ..... Phone: ..... (bh)

Address: .... (ah)

..... Net Address: .....

.....

..... *Write "Unchanged" if details have not*

..... *altered and this is a renewal.*

Student Member Certification *(to be completed by a member of the academic staff)*

I, ... certify that

..... *(name)*

is a full time student at ..... *(institution)*

and is expected to graduate approximately ____/____/____.

Title: ..... Signature: .....

Office use only:

Chq: bank _____ *bsb* _____-_____ *a/c* _____ *#* _____

Date: ___/___/___ *$*

Who: _____ *Memb#* _____

AUUG

Application for Institutional Membership
Australian UNIX* systems Users' Group.

*UNIX is a trademark of AT&T Bell Laboratories.

To apply for institutional membership of the AUUG, complete this form, and return it with payment in Australian Dollars to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

Foreign applicants please send a bank draft drawn on an Australian bank, and remember to select either surface or air mail.

..... does hereby apply for

☐ Renewal of existing Institutional Membership	$250.00
☐ New Institutional Membership of the AUUG	$250.00
☐ International Surface Mail	$ 20.00
☐ International Air Mail	$100.00
Total remitted	AUD$_____ (cheque, money order)

I agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

I understand that I will receive two copies of the AUUG newsletter, and may send 2 representatives to AUUG sponsored events at member rates, though I will have only one vote in AUUG elections, and other ballots as required.

Date: ___/___/___ Signed: _____

Title: _____

☐ Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

For our mailing database - please type or print clearly:

Administrative contact, and formal representative:

Name: ..... Phone: ..... (bh)

Address: .... (ah)

..... Net Address: .....

.....

..... *Write "Unchanged" if details have not*

..... *altered and this is a renewal.*

Please complete the other side.

Please send newsletters to the following addresses:

Name: .....
Address: .....
.....
.....
.....
.....

Name: .....
Address: .....
.....
.....
.....
.....

Write "unchanged" if this is a renewal, and details are not to be altered.

Please indicate which Unix licences you hold, and include copies of the title and signature pages of each, if these have not been sent previously.

Note: Recent licences usally revoke earlier ones, please indicate only licences which are current, and indicate any which have been revoked since your last membership form was submitted.

Note: Most binary licensees will have a System III or System V (of one variant or another) binary licence, even if the system supplied by your vendor is based upon V7 or 4BSD. There is no such thing as a BSD binary licence, and V7 binary licences were very rare, and expensive.

- ▫ System V.3 source
- ▫ System V.2 source
- ▫ System V source
- ▫ System III source
- ▫ 4.2 or 4.3 BSD source
- ▫ 4.1 BSD source
- ▫ V7 source
- ▫ Other *(Indicate which)* .....

- ▫ System V.3 binary
- ▫ System V.2 binary
- ▫ System V binary
- ▫ System III binary

Office use only:
Chq: bank _____ *bsb* ____-____ *a/c* _____ *#* _____
Date: ___/___/___ *$*
Who: _____ *Memb#* _____