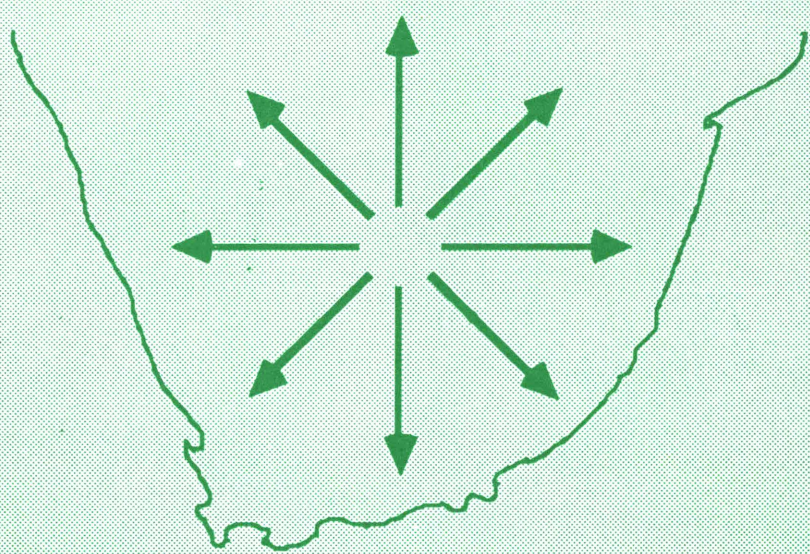
South African Computer Journal
Number 7
July 1992

Suid-Afrikaanse Rekenaar tydskrif
Nommer 7
Julie 1992

SPECIAL ISSUE



7th SA COMPUTER RESEARCH SYMPOSIUM

The South African Computer Journal

An official publication of the South African Computer Society and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif

'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
Email: dkourie@dos-lan.cs.up.ac.za

Subeditor: Information Systems
Prof John Schochot
University of the Witwatersrand
Private Bag 3
WITS 2050
Email: 035ebrs@witsvma.wits.ac.za

Productio.. Editor
Prof G de V Smit
Department of Computer Science
University of Cape Town
Rondebosch 7700
Email: gds@cs.uct.ac.za

Editorial Board

Professor Gerhard Barth
Director: German AI Research Institute

Professor Judy Bishop
University of Pretoria

Professor Donald Cowan
University of Waterloo

Professor Jürg Gutknecht
ETH, Zürich

Professor Pieter Kritzinger
University of Cape Town

Professor F H Lochovsky
University of Toronto

Professor Stephen R Schach
Vanderbilt University

Professor S H von Solms
Rand Afrikaanse Universiteit

Subscriptions

	Annual	Single copy
Southern Africa:	R45,00	R15,00
Elsewhere	$45,00	$15,00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

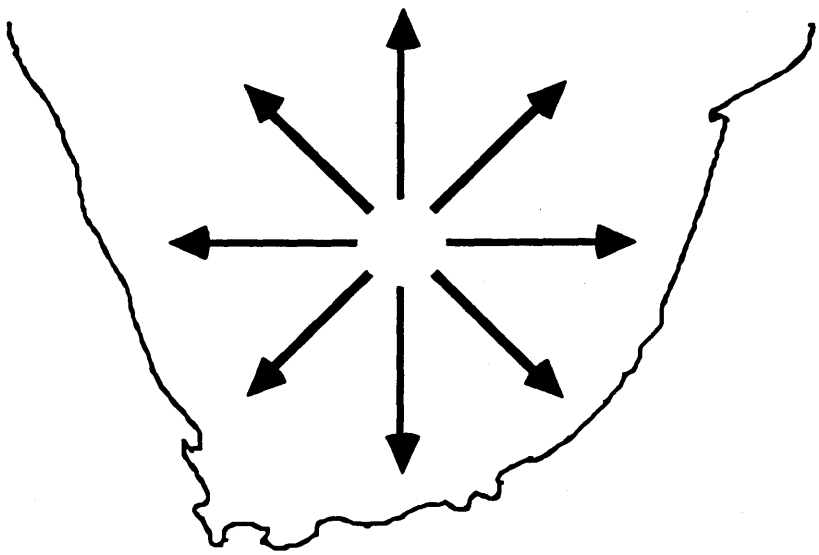
7th Southern African Computer Research Symposium

Karos Indaba Hotel, Johannesburg

1 July 1992

PROCEEDINGS

Guest Editor: Judy M Bishop



PERSETEL

Organised by the SA Institute of Computer Scientists
in association with the Computer Society of SA

Sponsored by Persetel and the FRD

SPECIAL ISSUE - 7th SA COMPUTER SYMPOSIUM

PREFACE

When the first SA Computer Symposium was held at the CSIR in the early eighties, it was unique. There was no other forum at the time for the presentation of research in computer science. In the intervening decade, conferences, symposia and workshops have sprung up in response to demand, and now there are several successful ventures, some into their third or fourth iteration. Each of these addresses a specific topic - for example, hypermedia, expert systems, parallel processing or formal aspects of computing - and attracts a specialised audience, well versed in the subject and eager to learn more. For the main part, the proceedings are informal, and certainly not archival.

SACRS, though, is still unique, in that it deliberately covers a broad spectrum of research in computing, and in addition, seeks to provide a lasting record of the proceedings. To achieve the second aim, we negotiated with the SA Institute of Computer Scientists for the proceedings to form a special issue of the SA Computer Journal, and the copy you have in front of you is the result. The collaboration between the symposium committee and the journal's editorial board placed high standards on the refereeing and final presentation of the papers, to the symposium's benefit, while we were still able to maintain a fresh, audience-oriented approach to the selection of papers.

This is SACJ's first such special issue, and the largest issue (at 145 pages) to date. We hope that it is only the beginning of future such collaborations.

In all 29 papers were received, all were refereed twice, and 19 were chosen for presentation by the programme committee. All the papers were thoroughly revised by the authors on the basis of the referee's comments, and the committee's suggestions aimed at making the material more accessible to a broadly-based audience. Papers had to be new, and not to have been presented elsewhere, a requirement that is still unusual within the SA conference round.

A third goal of SACRS has been to invite keynote speakers, usually from overseas. This year, we are fortunate to present Dr Vinton Cerf, the father of the Internet and a world-renown expert on computer networks. Although his paper is not available for this special issue, it will appear later in SACJ. Through the good offices of Professor Chris Brink of UCT, we also have three other speakers from Germany, Canada and the US adding interest to the event, and two of their papers appear in this issue.

The programme committee originally devised a theme for the symposium - "Computing in the New South Africa". We received several queries as to the meaning of this theme, but unfortunately few papers that addressed it directly. One prospective author went as far as to enquire whether computer research would survive in the new South Africa. Another felt that his work was definitely not in the theme, as it was genuine, old world, basic, theoretical science! Neverthless, there are two papers that consider one of South Africa's key issues, that of language. Others look at the success we have achieved in applying technology to mining, and the future of low-cost operating systems. In all, the mix of papers represents a balance between the theoretical and the practical, the past and the future, all firmly based in the computing of the present.

Organising the symposium has involved the hard work of several people, and I would like to thank in particular

- Derrick Kourie, my co-organiser, and the editor of SACJ for his invaluable advice and hard work throughout the planning and implementation stages;
- Riël Smit, the production editor, for attaining such a high standard in such a short time for so many papers;
- Gerrit Prinsloo and the staff at the CSSA for their efficeint and quite delightfully unfussy organisation;
- Persetel for their very generous sponsorship of R25000, and Tim Schumann for taking a genuine interest in our events;
- the Foundation for Research Development for sponsoring Vint Cerf's visit;
- and finally the Department of Computer Science of the University of Pretoria for providing the ideal working conditions for undertaking ventures of this kind, and especially Roelf van den Heever for his unfailing encouragement and support.

Judy M Bishop
Organising Chairman, SACRS 1992
Guest Editor, SACJ Special Issue

Referees

The journal draws on a wide range of referees. The following were involved in the refereeing of the papers selected for this special issue. Their role in certifying the papers and their contribution to enhancing the quality of papers is sincerely appreciated.

John Barrow	UNISA
Ronnie Becker	University of Cape Town
Danie Behr	University of Pretoria
Sonia Berman	University of Cape Town
Liesbeth Botha	University of Pretoria
Theo Bothma	University of Pretoria
Chris Brink	University of Cape Town
Peter Clayton	Rhodes University
Ian Cloete	Stellenbosch University
Antony Cooper	CSIR
Elise Ehlers	Rand Afrikaanse University
Quintin Gee	DALSIG
Andy Gravell	University of Southampton
Wendy Hall	University of Southampton
David Harel	The Weizmann Institute of Science
Scott Hazelhurst	University of the Witwatersrand
Derrick Kourie	University of Pretoria
Willem Labuschagne	UNISA
Doug Laing	ISM
Dave Levy	University of Natal
Graham McLeod	University of Cape Town
Hans Messerschmidt	University of the Orange Free State
Deon Oosthuizen	University of Pretoria
Ben Oosthuysen	University of Pretoria
Neil Pendock	University of the Witwatersrand
D Petkov	University of Natal
Martin Rennhackkamp	Stellenbosch University
Cees Roon	University of Pretoria
Jan Roos	University of Pretoria
John Shochot	University of the Witwatersrand
Morris Sloman	Imperial College, London
Riel Smit	University of Cape Town
Pat Terry	Rhodes University
Walter van den Heever	University of Pretoria
Lynn van der Vegte	University of Pretoria
Herman Venter	University of Fort Hare
Herna Viktor	Stellenbosch University

A Multitasking Operating System above MS-DOS

R J Foss G M Rehmet R C Watkins
Department of Computer Science, Rhodes University, Grahamstown 6140
email: csrf@giraffe.ru.ac.za

Abstract

This paper describes the development of a memory resident version of the Xinu operating system for the IBM PC range of computers. The operating system co-resides with the MS-DOS operating system. The need for the operating system is motivated in the context of real-time distributed system development. Important features of the Xinu operating system are discussed. The paper shows how these features are made available to programs developed under the widely used MS-DOS operating system.
Keywords: *Multitasking, MS-DOS, Xinu, Distributed Applications.*
Computing Review Categories: *D.4.1, C.2.4.*

Received February 1992, Accepted April 1992

1 Introduction

The Xinu operating system is a multitasking operating system with a layered design. The operating system has a number of UNIX-like features, but is by no means a UNIX clone. Indeed, Xinu stands for 'Xinu is not UNIX'! Xinu, originally developed on LSI-11 based workstations, has been ported to a number of platforms. Our work has focused on the IBM PC version of the operating system. This paper describes the conversion of the IBM PC version into a memory resident format, where it co-resides with the MS-DOS operating system.

The motivation for the creation of the operating system stemmed from a particular project within the Computer Science Department at Rhodes University. However, it will be apparent that the characteristics required of the operating system in this project are needed in many other application areas. The project involves the development of a large distributed system with real-time characteristics. The system comprises a network of music workstations which all access central music studio resources, such as synthesizers and tape recorders. A major goal of the project is to make the potential of computer-based studios more easily and widely accessible to South African musicians.

The music studio resources are all under the control of a server. The server controls the booking of resources and the patching of resources to the various workstations. The workstations need to transmit musical information to the resources in real time. They must also transmit patching and audio mixing messages to the server in real-time. The server must deal with a number of patching and mixing requests from the various workstations in real time. A more detailed description of the system can be found in [8].

The system was analyzed using the real-time structured analysis tools of Ward and Mellor [13], as well as the object oriented data analysis tools of Shlaer and Mellor [11]. The initial analysis clearly indicated that the implementation of the system would be difficult without the facilities of a multitasking operating system with preemptive scheduling. The operating system should allow for the easy incorporation of drivers for a range of peripheral devices. These included RS232 compatible tape drives and devices with the Musical Instrument Digital Interface (MIDI). Communication across an ethernet network was also essential.

The hardware available for implementation was the IBM-PC range of computers. This included those with 8088 processors. However, it would be most desirable if the code could be ported to other hardware platforms. It particular, the server could be replaced by a SUN workstation and the workstations by Apple Macintosh's.

2 Available Operating Systems

There did not appear to be any commercial operating systems which fitted all the requirements of the application. MS-DOS, the original and most widely used IBM-PC operating system, does not allow the creation of multiple, simultaneous tasks. UNIX is a general purpose operating system, and its designers did not intend it for real time use. The effects of background processing in the XENIX system V operating system have been described in [14]. Both OS/2 and Windows 3.0 require at least an 80286 processor with substantial memory to run effectively. This is also true of most UNIX systems running on IBM PC's. Windows 3.0 does not perform preemptive rescheduling, and does not allow for the spawning of child processes [10].

On a much smaller scale, a number of multitasking kernels have been developed for IBM PC implementations of Pascal, C and C++ [9, 12]. These kernels tend to provide only cooperative multitasking facilities. This approach avoids the reentrancy problem posed by MS-DOS system routines. Also, these kernels do not typically provide a complete, congruent I/O system, which allows for the easy incorporation of device drivers. Usually, they are written

with a specific application in mind.

Commercial, IBM-PC based compilers exist for Modula-2, a language which incorporates a coroutine facility. Runtime libraries exist for these compilers which employ coroutines and interrupts for the creation of multiple time-sliced processes. An example is the QuickMod library set developed for the Stony Brook compiler [6]. The modules provided are modelled on the 'Processes' module described by Wirth [15]. While the process creation and synchronization facilities are well-developed, all I/O depends entirely on DOS. Rather coarse mutual exclusion primitives have to be applied to get around the lack of reentrancy in MS-DOS. An I/O system would have to be developed for non-standard devices, or DOS device drivers would have to be created for them.

A further factor in the choice of operating system was the cost of the software. Each workstation on the network would have to have a copy of the operating system. An expensive operating system requiring expensive hardware resources would make the computer music network out of reach for most school, college, and university music departments.

3 The Xinu Operating System

For some years, the Xinu operating system has been used in the Computer Science Department at Rhodes University as a vehicle for teaching operating systems [7]. Xinu was created by Douglas Comer, who was working at the time with Bell Laboratories. The original version of Xinu was developed for a number of diskless, networked, LSI-11 workstations. The source code for this version of the operating system appeared in text book form [2]. Xinu has now been ported to a number of other platforms, including SUN workstations and the Apple Macintosh. Timothy Fossum of the University of Wisconsin ported it to the IBM PC and described this version of the operating system in a text book written in collaboration with Douglas Comer [5].

The Xinu operating system is designed in layers and has all the essential features of a multitasking operating system. It allows for the creation of processes with varying priorities. These processes run concurrently, with the highest priority, eligible process always having control of the hardware processor. Equal priority processes are scheduled on a round-robin basis. Process coordination primitives are provided in the form of semaphores. There are two types of message passing schemes, with associated primitives. The first allows for the transmission and receipt of single messages, the second for the creation of ports with associated message queues. A memory manager allows for the creation and freeing of memory blocks. Memory can also be retrieved from buffer pools.

Input and output to devices is performed via a small set of high level system calls. Device switch tables, established at configuration time, map these calls to specific device drivers. The input/output system is conceptually simple and it is easy to incorporate drivers for new devices. A simple non-overlapping window system has been developed for PC-Xinu, and is described in [5].

Comer has written two further books which describe the implementation of internetworking facilities in the context of the Xinu operating system [3, 4]. Code which implements the User Datagram Protocol suite has been added to the IBM PC version of Xinu.

4 The Development of Programs for PC-Xinu

The PC-Xinu operating system, as developed by Fossum [5], comprises a library of functions which can be linked to a user program. The user program is written as a Xinu program and is developed using tools which run under MS-DOS. In our case, these tools comprise the Borland Turbo C editor and compiler and the Microsoft 'make', 'link', and assembler utilities. An MS-DOS executable program file is created. When run, this program initializes the Xinu operating system and starts up the main user process. This process can then spawn a number of other processes using Xinu system calls. Effectively, the user program runs under Xinu. The MS-DOS loader is used to load up a combination of the Xinu operating system and user program.

Under this system, every Xinu program comprises a copy of the operating system together with the user program running 'under' the operating system. This has a number of disadvantages. Firstly, linking can be time consuming and this is not conducive to quick development. The operating system occupies the same memory space as the user program. If the user is developing under the small memory model, where only 16 bit offset addresses are used for data and function access, Xinu will occupy a substantial proportion of the overall memory space. The small memory model provides speed advantages over the large memory model, where both segment and offset addresses are used for function and data access. A more debatable disadvantage stems from the fact that all operating system data structures are available for manipulation by user processes. Sometimes, there are advantages to viewing these structures, but generally user processes should simply be presented with a 'system call' interface to the operating system. Lastly, all Xinu programs developed under this environment will have large executable files associated with them, since every program file comprises the user program and the Xinu operating system.

5 The creation of a 'memory resident' PC-Xinu

In order to overcome the above problems associated with PC-Xinu development, it was decided to separate the operating system from the user program. The MS-DOS operating system incorporates a system call which allows a program to remain resident in memory after it terminates execution. A number of commercial packages have utilized this system call to provide a multi-program desktop environment under MS-DOS. Turbo C provides an easy-to-use

interface to this system call. Functions in such a resident program can later be accessed via hardware or software interrupts.

In order to create the memory resident operating system, a small program has been written and linked to the Xinu library. This program incorporates the system call dispatcher, a revised Xinu initialization function, and code to make both itself and the operating system resident in memory. The code below provides the essentials for making the operating system resident and accessible to user programs.

```
setvect(UserInt, Syscall); /* dispatcher vector*/
sizeprogram = _DS-_psp+0x1000; /* Xinu size    */
keep(0, sizeprogram);       /* make resident   */
```

In the first line of the above code, UserInt is a software interrupt number, and Syscall is the address of a dispatcher function which will dispatch control to any one of a number of system functions. The `setvect` call allows a user to call the dispatcher via a software interrupt. The second line calculates the size of the Xinu operating system in terms of 16-byte paragraphs. `_DS` is the 80x86 data segment register, and `_psp` is the program segment prefix, indicating the start of the MS-DOS program in memory. The third line contains the Turbo C library function call to make the operating system resident.

On the user side, every user program running under the Xinu operating system must be linked to a small library which converts standard Xinu system calls into corresponding software interrupts. Thus, for example, the Xinu system call to create a new process would typically be used in the following way:

```
create(procaddr,ssize,priority,namep,
       nargs,arg1,arg2,..);
```

where `procaddr` is the code to be executed by the new process, `ssize` is the stack size allocated to the new process, `priority` is the priority of the new process, `namep` is a symbolic name for the process, `nargs` is the number of arguments to be passed to the process, `arg1,arg2,..` could be one or more arguments passed to the process.

This function is coded in the small Xinu user library as:

```
create()
{
  asm mov ax,5      /* the 'create' system call */
  asm int UserInt   /* dispatcher interrupt     */
}
```

The code makes use of the in-line assembly features of the Turbo C compiler. The 80x86 AX register is assigned an indicator value. The dispatcher in the memory resident Xinu operating system will use this value to call the `create` system function.

It is clear from the above code that none of the parameters for the `create` system call are placed in registers. These parameters reside on the stack of the calling process. The resident operating system functions have been coded in such a way that they pick these parameters up from the stack. This prevents any speed degradation resulting from the loading of parameters into registers. Of course the parameters must be pushed on to the stack according to C language conventions. If the resident Xinu system functions are to be utilized via a different language, the C parameter passing conventions will have to be adhered to. A similar sort of convention must be applied under Microsoft Windows, where MS-DOS compilers are also used for development. Under Windows, however, the Pascal parameter passing conventions are used [10].

Before the main user process is executed, the operating system is initialized. The initialization process involves firing up various important system processes, particularly those that handle input/output, and then handing control over to the main user process. This initialization process is activated via a software interrupt and is hidden from the application programmer.

6 Problems associated with the resident Xinu

Most of the problems associated with creating the resident Xinu had to do with the segmented memory of the IBM PC. This, in turn, stems from the architecture of the 80x86 processors. The Xinu source code as written by Comer and Fossum assumes that operating system and program share the same address space. In the IBM PC resident version, the operating system resides in code and data segments which are quite separate from the user program's segments. These problems are compounded by the fact that both user programs and operating system are written using the small memory model compiler of Turbo C. Under this compiler, all code and data access is via offset addresses. If these offset addresses are passed as parameters to the resident operating system, it will not be able to access the address contents, unless it also knows the associated segment. To complicate matters further, each process has its own stack area on which parameters and local variables are stored. This stack area is allocated from the operating system's data area.

In the resident PC-Xinu all address parameters are passed as far pointers. This means that both offset and segment addresses are passed. System functions have been prototyped in such a way that this happens automatically. System functions in the resident operating system have been altered to expect these far pointers.

The Xinu context switcher must be able to start up either operating system processes in the current code segment, or user processes in a different code segment. This difficulty has been overcome by setting up the new process' stack in such a way that the context switcher switches to an operating system function. This function reloads the user data segment and performs a far return to the new user process. This implies that the operating system 'knows' what the user data and code segments are. The code for this function comprises a single line and is given below.

```
void far getback()
{
   asm  pop ds
}
```

A number of the above segmentation problems have been

overcome by using a large memory model for the creation of both operating system and user programs. Under this model, address parameters automatically comprise segment and offset values. This model also allows for the complete utilization of the IBM-PC 640k usable address space by the operating system and user program. There will, of course, be some speed degradation under the large memory model.

7 Development under the resident PC-Xinu operating system

The Xinu operating system is loaded by executing the memory resident program described above. A Xinu program is edited and compiled using standard MS-DOS tools. It is then linked to the required language-specific standard libraries, and a small Xinu library which furnishes the necessary software interrupts. Using Turbo C, this can all be done within the Turbo C 'integrated environment' [1]. The Xinu operating system can be removed from memory by running the same memory resident program and requesting removal.

8 Future Developments

Currently, the resident Xinu operating system comprises a process manager, a memory manager, process coordination and message passing facilities, a Xinu file system, an MS-DOS file system, and a text-based, non-overlapping window system. Programs can be developed under the small or large memory model of Turbo C. Of course, development can take place in other environments, as long as the parameter passing conventions of the operating system are adhered to.

A basic overlapped windowing system has been developed and this is being extended to provide a Microsoft Windows-like programming environment. Thus, the operating system will capture user-generated events, typically mouse button and keypress selections, and will send them to an associated process.

As mentioned above, Comer has provided the complete source code for a suite of UDP and TCP/IP functions. This code will be added to the memory resident operating system on top of low-level ethernet drivers, which have already been developed in the Computer Science Department at Rhodes.

9 Conclusion

The IBM PC in its various forms is an inexpensive and easily available hardware platform. A vast range of development tools have been developed for use under its original operating system, MS-DOS. A large number of programmers are familiar with these tools, particularly in South Africa where IBM PC's pervade the universities and technikons. However, the operating system with its lack of process management and coordination facilities does not easily lend itself to the development of real-time or distributed systems.

This paper has described how the multi-process facilities of the Xinu operating system have been made easily accessible to programmers developing programs within the MS-DOS environment. Source code for the operating system is available, enabling a thorough understanding of the various system functions. A clean, congruent input/output system provides a template for the addition of further device drivers. There is a complete, well-documented UDP and TCP/IP implementation for Xinu. All this means that programmers can develop sophisticated real-time distributed applications for the IBM PC range of computers without investing the time and capital for a new development environment.

References

1. Borland International Inc. *The Turbo C/C++ User's Guide*, 1987–1990.
2. D Comer. *Operating System Design, the Xinu Approach.* Prentice-Hall, 1983.
3. D Comer. *Operating System Design, Volume 2, Internetworking with Xinu.* Prentice-Hall, 1987.
4. D Comer. *Internetworking with TCP/IP, Volume 2, Design Implementation and Internals.* Prentice-Hall, 1991.
5. D Comer and T Fossum. *Operating System Design, Volume 1, the Xinu Approach.* Prentice-Hall, IBM PC edition, 1988.
6. D Cooper. *Using QuickMod.* W W Norton & Company, New York and London, 1990.
7. R J Foss. 'Teaching operating systems using the IBM PC'. In *Proceedings of the 17th SACLA Conference*, Pretoria, (1987).
8. R J Foss and A Wilks. 'A network approach to the sharing of music studio resources'. In *Proceedings of the 1990 International Computer Music Conference*, Glasgow, (1990).
9. C A Lindley. 'Multitasking with Turbo Pascal'. *Dr. Dobb's Journal*, p. 129, (1987).
10. C Petzold. *Programming Windows.* Microsoft Press, 1990.
11. S Shlaer and S Mellor. *Object Oriented Systems Analysis – Modeling the World in Data.* Prentice-Hall, 1988.
12. M Tarpenning. 'Cooperative multitasking in C++'. *Dr. Dobb's Journal*, (1991).
13. P Ward and S Mellor. *Structured Development for Real-Time Systems*, volume 1, 2 and 3. Yourdon Press, 1986.
14. G C Wells. 'An evaluation of Xenix System V as a real-time operating system'. *Microprocessing and Microprogramming*, 33:57–66, (1991).
15. N Wirth. *Programming in Modula-2.* Springer-Verlag, New York, 1983.

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications or Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript

Manuscripts for *review* should be prepared according to the following guidelines.

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be numbered and titled. Figures should be submitted as original line drawings/printouts, and not photocopies.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text in square brackets [1, 2, 3]. References should take the form shown at the end of these notes.

Manuscripts accepted for publication should comply with the above guidelines (except for the spacing requirements), and may be provided in one of the following formats (listed in order of preference):

1. As (a) LaTeX file(s), either on a diskette, or via e-mail/ftp – a LaTeX style file is available from the production editor;
2. In camera-ready format – a detailed page specification is available from the production editor;
3. As an ASCII file accompanied by a hard-copy showing formatting intentions:
   - Tables and figures should be on separate sheets of paper, clearly numbered on the back and ready for cutting and pasting. Figure titles should appear in the text where the figures are to be placed.
   - Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.

   Further instructions on how to reduce page charges can be obtained from the production editor.
4. In a typed form, suitable for scanning.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for LaTeX or camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers in categories 3 and 4 above will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to less than about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

References

1. E Ashcroft and Z Manna. 'The translation of 'goto' programs to 'while' programs'. In *Proceedings of IFIP Congress 71*, pp. 250–255, Amsterdam, (1972). North-Holland.
2. C Bohm and G Jacopini. 'Flow diagrams, turing machines and languages with only two formation rules'. *Communications of the ACM*, 9:366–371, (1966).
3. S Ginsburg. *Mathematical theory of context free languages*. McGraw Hill, New York, 1966.

South African Computer Journal

Number 7, July 1992
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 7, Julie 1992
ISSN 1015-7999

Contents