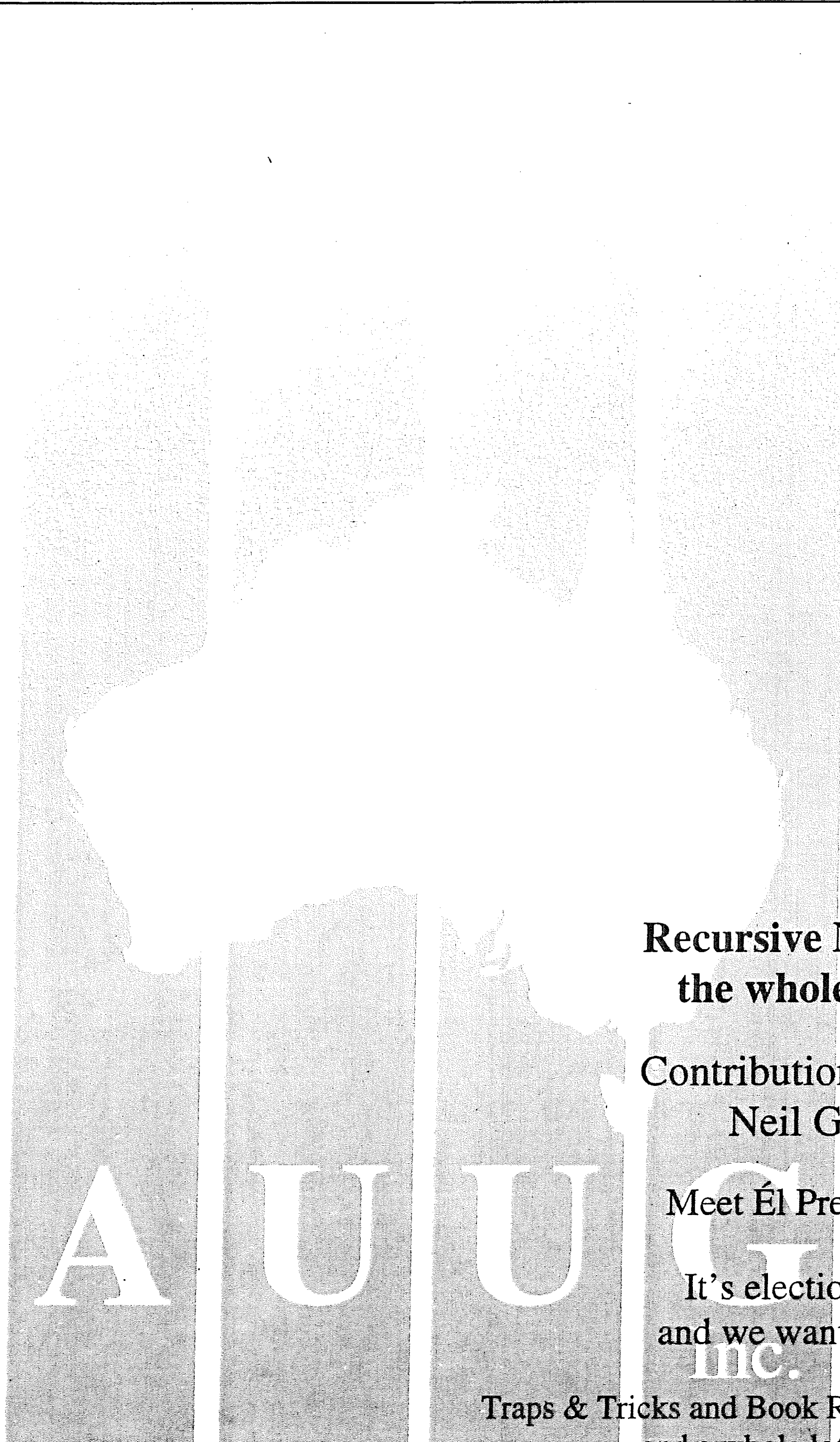
AUUGN

The Journal of AUUG Inc.
Volume 19 • Number 1
February 1998



UNIX & OPEN SYSTEMS USERS

ISSN 1035-7521

Print post approved by Australia Post - PP2391500002

AUUGN

The Journal of AUUG Inc.

Volume 19 • Number 1
February 1998

Editorial	3
President's Column	3
NT to the Max...(NoT)	6
The ABC's of TPC's and NT Scalability II	8
Recursive Make Considered Harmful	14
Book Reviews	27
Meet the AUUG-Exec	29
Call For Papers	32
SAGE-AU Sixth Annual Conference and General Meeting	33
AUUG Incorporated 1998 Annual Elections Call for Nominations	37
AUUG Incorporated Election Procedures	37
Returning Officer's Report AUUG Rules Ballot, September 1997	40
Chapter News: Canberra	40
Chapter News: AUUG-NSW	40
AUUG Local Chapter Meetings 1998	41
UNIX Traps & Tricks	42

Thanks to our Sponsors:

digital™

TELLURIAN

AUUG Membership and General Correspondence

The AUUG Secretary
PO Box 366
Kensington NSW 2033

Tel: 02 9361 5994
Fax: 02 9332 4066
Toll Free: 1800 625 655
Internet: *auug@auug.org.au*

AUUG Executive

President:
Michael Paddon
Michael.Paddon@auug.org.au
Australian Business Access
723 Swanson Street
Carlton VIC 3053

Vice President:
Lucy Chubb
Lucy.Chubb@auug.org.au
Softway Pty. Ltd.
79 Myrtle St
Chippendale NSW 2008

Secretary:
David Purdue
David.Purdue@auug.org.au
Sunsoft Pacific
Level 4, Sunsoft Building
44 Hampden Road
Artarmon NSW 2064

Treasurer:
Stephen Boucher
Stephen.Boucher@auug.org.au
MTIA
509 St. Kilda Road
Melbourne VIC 3004

Committee Members:
Malcolm Caldwell
Malcolm.Caldwell@auug.org.au
Northern Territory University
Casuarina Campus
Darwin NT 0909

Luigi Cantoni
Luigi.Cantoni@auug.org.au
STM
Suite 3, 77 Millpoint Road
South Perth WA 6153

Peter Laytham
Peter.Laytham@auug.org.au
SCO
Level 7, 157 Walker Street
North Sydney NSW 2060

Mark White
Mark.White@auug.org.au
Tandem Computers
143 Coronation Drive
Milton QLD 4064

AUUG Business Manager:

Elizabeth Egan
busmgr@auug.org.au
Level 4, 90 Mount Street
North Sydney NSW 2060

Contribution Deadlines for AUUGN in 1998/99

Volume 19 • Number 2 • May 1998 :	**April 7th 1998**
Volume 19 • Number 3 • August 1998 :	**July 7th 1998**
Volume 19 • Number 4 • November 1998 :	**October 7th 1998**
Volume 20 • Number 1 • February 1999 :	**January 7th 1999**

AUUGN Editor

Günther Feuereisen
gunther@ibm.net
PO Box 366
Kensington NSW 2033

AUUGN Correspondence

Please send all correspondence regarding AUUGN to:

AUUGN Editor
auugn@auug.org.au
PO Box 366
Kensington NSW 2033

Submission Guidelines

Submission guidelines for AUUGN contributions are regularly posted on the **aus.org.auug** news group.

They are also available from the AUUG World Wide Web site at:

http://www.auug.org.au

Alternately, send email to the above correspondence address, requesting a copy.

AUUGN Back Issues

A variety of back issues of AUUGN are still available; for price and availability please contact the AUUG Secretariat, or write to:

AUUG Inc.
Back Issues Department
PO Box 366
Kensington NSW 2033
Australia

Conference Proceedings

A limited number of copies of the Conference Proceedings from previous AUUG Conferences are still available. Contact the AUUG Secretariat for details.

Mailing Lists

Enquiries regarding the purchase of the AUUGN mailing list should be directed to the AUUG Secretariat.

Tel: (02) 9361 5994
Fax: (02) 9332 4066

During normal business hours.

Disclaimer

Opinions expressed by the authors and reviewers are not necessarily those of AUUG Inc., its Journal, or its editorial committee.

Copyright Information

Copyright © 1998 AUUG Inc.
All rights reserved.

AUUGN is the journal of AUUG Inc., an organisation with the aim of promoting knowledge and understanding of Open Systems, including, but not restricted to, the UNIX® operating system, user interfaces, graphics, networking, programming and development environments and related standards.

Copyright without fee is permitted, provided that copies are made without modification, and are not made or distributed for commercial advantage.

Editorial

Günther Feuereisen <gunther@ibm.net

Happy 1998 Everyone!!

I trust everyone had a good break, and we're all ready to go once more!

The holidays are generally a quiet time, but there has been some noise in the computer world. On January 26th, Digital and Compaq announced a US$9 billion deal, which effectively sees Digital become part of Compaq. This follows a deal by Compaq in late 1997 which saw them acquire Tandem.

What does this mean? Well, at the moment there seems to be a lot of speculation, but you need to look at what Compaq has been buying.

Tandem and Digital are world leaders in High Availability, or Cluster technology products. By acquiring both companies, Compaq now has the biggest base of UNIX cluster technology, as well as picking up VMS clusters from Digital, and combining it with NT clusters which Compaq has already been providing.

Compaq is now, in effect, a one stop cluster house for all your High Availability needs, irrespective of your corporate server environment.

Definitely something to rattle the other HA players out there.

This should provide interesting times ahead in the HA server market, it will be interesting to see what happens with the evolution of the various products; if they stay as separate entities, or merge to some common environment.

I hope everyone is enjoying the summer conferences, as well as the roadshows!

See you in May!

❖

President's Column

Michael Paddon <Michael.Paddon@auug.org.au>

Netscape Communications Corporation finally worked out what the rest of us knew already. Free software wins every time. And it has been obvious to all observers for quite a while that Microsoft's Explorer was systematically stomping Mozilla into a green, lizard flavoured pate. Something was going to have to be done, or the wunderkind of the corporate internet industry was going to make a spectacular exit from the history books.

And, lo!, something was done. On January 22, our favourite reptile announced a "bold move" to "harness the creative power ... [of] developers". More precisely, Navigator was now free for all to use and enjoy. Let's quietly ignore the fact that most users weren't paying for it anyway, and that this could be seen as a face saving realisation that they never would. By any yardstick, this was a radical move relative to traditional corporate thinking: telling your shareholders that you're now giving away your flagship product isn't exactly plain sailing.

One is left wondering, however, whether or not our saurian buddy has really got the point. You see there is a difference between free software and well, ummm, stuff you can download from the net. And, no, I'm not talking about the cost of bandwidth here.

Truly free software is code that people can use, understand, modify and pass on without restriction. Money has nothing to do with it. If you want to charge me one dollar or a million for a piece of software, or even give it to me for nothing, that is your right. I get to decide whether or not I am getting good value. If you try and charge me a thousand bucks for
a web server I'll probably just go and download Apache. On the other hand, if you ask thirty for an OpenBSD CDROM, I just might decide that it is well worth my hard earned cash.

I'm not saying that software isn't worth money, or that we shouldn't get paid for investing and working with it. I'm just saying that the ways we measure that worth and how the fiscal rewards flow from it are changing. This sort of change is common in history. Reflect on the economics of the publishing industry pre and post printing press to see what massive changes technology has wrought.

OK, this is a radical idea, though hardly original. But let's work with it for a while.

If a piece of software is free, then the dynamics of the internet can go to work. Millions of beta testers (you can think of them as users), thousands of developers, hundreds of insights. I've played this tune before, and

it suffices to mention a few examples like Linux, the GNU project, the BSD variants and the XFree consortium to prove that free software kicks butt. Whatever happened to NeWS, OSF, DCE, AFS and NextStep, to name a few? All great technologies and, sure, a few still languish in commercial obscurity, but they didn't exactly change the world now, did they? Instead, they seemed caught in some kind of go slow time warp while the rest of the world accelerated away (some may still be resurrected as free software, with luck).

When you think about the modern software economic in these terms, it throws the way we think about open systems into question. A lot of people have tried to convince us that "open" meant standards conformant or, at least, interoperability. In the end, however, they wanted to ship us chunky, monolithic software that felt more like a straitjacket than a tool. Standards and interoperability are good, but they are now baseline expectations like reliability and ease of use.

As we head towards the next century our real challenge is not going to be fixing some stupid COBOL program with two digit year fields, but rather redefining the meaning of open to include and require the freedoms mentioned above. We have no choice... we want more from our software than ever before, and our demands grow by an order of magnitude each decade. The old economic model cannot support this, and it's now time to evolve or, well, face the fate of the dinosaurs.

Which brings us back to our pal Mozilla. There's nothing "bold" about making a web browser free to use. And if he (or she... we only get a head and shoulders shot on the browser icon so it's hard to tell) thinks that the "creative power" will be "harnessed" by plonking a monolithic slab of software in our backyards, with HTML as the only open interface, then I suggest that looking for alternative employment may be wise. Laying waste to Tokyo is always a career option for large, cold blooded bipeds.

If I can't open the black box and make it work better, then I can't tailor it to my needs in ways that only I can foresee. If I can't hook it up with my other software efficiently and elegantly there is no synergy between the components of my system. If code is not free and open (and I submit that for important values of "free", this is a redundant phrase) then I can't be competitive and I can't realise my ideas into working software.

The days of writing everything from scratch are over. Modern software is the most complicated machinery on the planet and we need to build on each other's successes to continue forward. Openness and freedom have everything to do with this.

Netscape may have, indeed, understood this. In the same press release they committed to "post the source code for free for Communicator 5.0". Now that is beyond radical. That is a twenty first century business model. And if they follow through on this commitment, if they make the release truly free and open software, then we are going to see Mozilla come back with a vengeance. Forget Tokyo... Redmond is going to be wasted, and Bill Gates will have a lot more to worry about than a cream pie in the face.

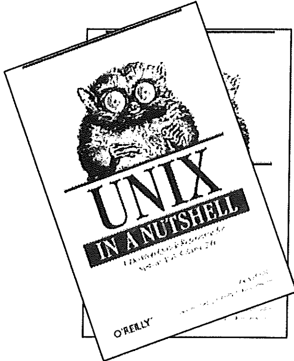
Go, Go, Mozilla.

Want to know more about AUUG?

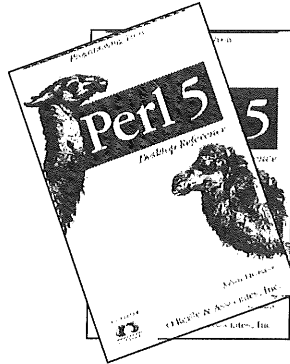
Check our the AUUG website for more information:

www.auug.org.au

O'REILLY ANIMAL MAGNETISM

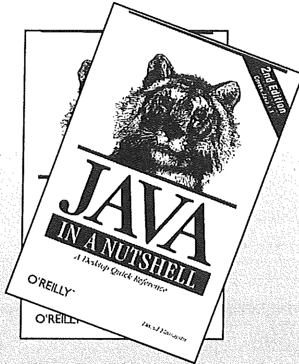


UNIX In a Nutshell System V & Solaris 2 2/e
Gilly ORE
1565920015 19.95

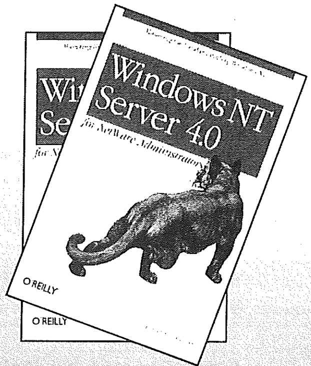


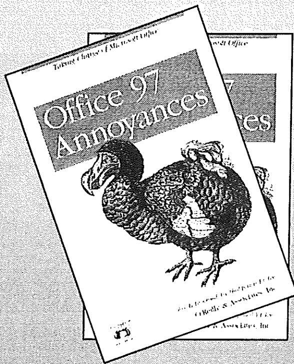
Perl 5 Desktop Reference
Vromans ORE
1565921879 9.95

Java in a Nutshell 2/e
Flanagen ORE
156592262X 39.95

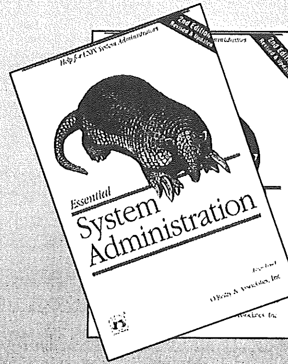


Windows NT Server 4.0 for NetWare Administrators
Thompson ORE
1565922808 79.95





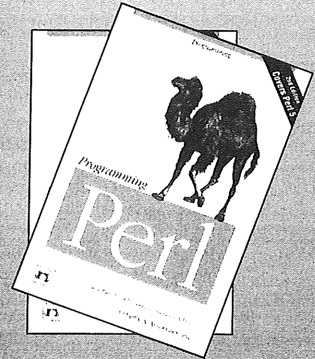
Office 97 Annoyances
Leonhard ORE
1565923103 44.95

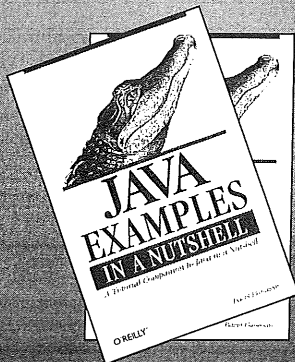


Essential System Administration 2/e
Frisch ORE
1565921275 65.00

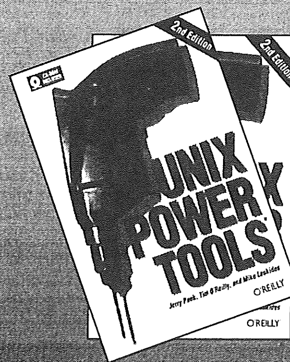
TCP/IP Network Administration 2/e
Hunt ORE
1565923227 65.00

Programming Perl 2/e
Wall ORE
1565921496 79.95





Java Examples in a Nutshell
Flanagen ORE
1565923715 39.95



Unix Power Tools 2/e (Bk/CD)
Peek ORE
1565922603 120.00

*rrp = reccomended retail price; note price and availability subject to change without notice

Available from all good bookstores
OR
direct from woodslane on
Tel: (02) 9970 5111 Fax: (02) 9970 5002

O'REILLY™

NT to the Max...(NoT)

Neil Gunther <ngunther@ricochet.net>
© 1997 Performance Dynamics. All Rights Reserved.

To NT or NoT to NT? As someone who analyzes the performance of large-scale UNIX servers, I finally decided to face this nagging question by attending the recent USENIX WindowsNT Workshop in Seattle. Seattle's weather being in the 90's all week was as much of a surprise as having the trade names USENIX and NT in the same conference title! Surprises, however, turned out to be a theme of the conference. Overall, I came away being much more impressed with NT than I had expected and I'm very glad I attended. But there were some unexpected low points too and it's one of those I would like to bring to your attention. It has do with a topic near and dear to the hearts of many UNIX SysAdms and others involved in managing large UNIX systems--server scalability [1]. It is also the next frontier for Microsoft.

Naturally, one presentation I was looking forward to was the opening address by Jim Gray entitled, "Windows NT to the Max—Just How Far Can It Scale Up?" Gray is a respected figure in the database community who's career spans companies like Tandem, DEC, and now Microsoft. Moreover, he's possibly best known as one of the leading evangelists [2] for standardizing database benchmarks that ultimately led to the formation of the Transaction Processing Performance Council.

Imagine my surprise to hear from Gray that NT could outperform UNIX by scaling to billions of transactions at prices much lower than UNIX platforms. Imagine my *dismay* at some of the subtle misinformation invoked to make the point! OK. You can't imagine my dismay: either you weren't there or the details were unfamiliar to you. Either way, I would like to examine some points in Gray's presentation with a view to separating the technical "wheat" from the marketing "chaff." Since this will require excursions off the beaten UNIX path, more details will appear in forthcoming `;login:` articles. In this opening salvo, there is only space to highlight the issues I plan to revisit. Eventually, I hope this discussion will better help you understand the scalability of both NT and UNIX.

What's Wrong With This Picture?

So, what did the man say? Figure 1 represents OLTP (On-Line Transaction Processing) throughput using the TPC Benchmark™ C. The TPC-C benchmark workload models inventory control in a distributed warehouse, and performance is measured in transactions per minute (or tpmC) [3]. The benchmark must be audited by the TPC and documented before being announced publicly, otherwise it's a technical violation of TPC rules.

Figure 1 appeared in Gray's presentation with the title: "NT Scales Better Than Solaris." Although not a usage violation of TPC data, the reader needs to exercise extreme caution when reading comparative charts of this type. The major variables in any TPC benchmark are:

- Platform (e.g., Intel or Sun processors and disks)
- Operating system (e.g., NT or Solaris)
- RDBMS (e.g., database management software; SQLServer or Sybase)
- Application (e.g., the TPC-C benchmarking workload or other)

The performance analyst's golden rule is: Only change one thing at a time [1]. In Fig. 1 there are many things that are different.

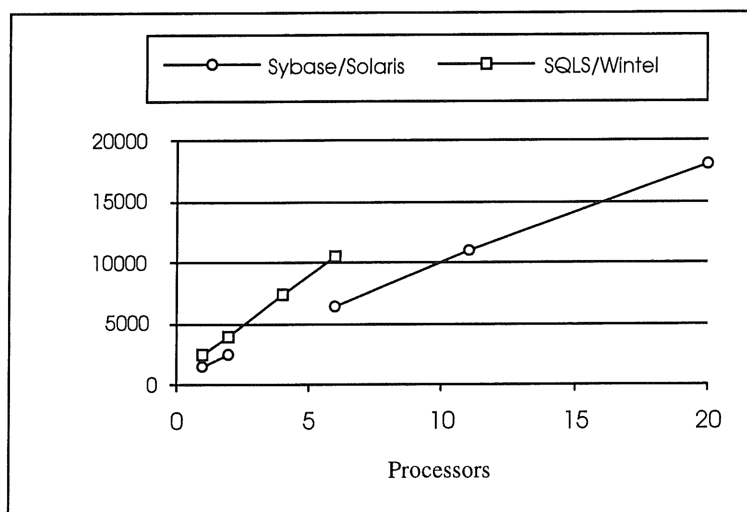


Figure 1. Comparative server scalability (no 4-way result was published by Sun).

Here are some variables to watch out for:

- Are all SQLServer data points on the **same** wintel architecture?
- SQLServer and Sybase are not the same RDBMS.
- Are there any Sybase results on wintel platforms, and how do they compare?
- Microsoft has full control over SQLServer code and its performance.
- Why are there no TPC-C data points above a 6-way wintel server?
- Do other RDBMSs scale better than SQLServer on wintel platforms?
- How contemporaneous were these measurements?

Incidentally, Gray publicly blamed Intel-based hardware for currently limiting scalability performance above 6 way servers. We'll return to these points in a subsequent article.

Billions and Billions!

To give some idea of the future potential of large-scale NT servers Gray reported on several scalability projects:

- Online Atlas (1.1 Million US place names with SPIN-3 images)
- Tandem's "Two-Ton" (DSS workload on 64 CPUs with 2 TBytes on 480 disks)
- Compaq "Debit-Credit" (OLTP workload on 140 CPUs and 900 disks)

The last of these projects apparently supports over 1 billion database transactions per day. These large-scale prototype systems are non-trivial to construct and the results in themselves are very impressive. Note that 1 billion transactions per day is about 700,000 transactions per minute. But Microsoft has apparently entered the Carl Sagan zone of benchmarking because these are *not* TPC-C transactions or any other TPC benchmark transactions. Why not? Since Gray did not explain this point to the conference audience, I asked him about it afterward. He told me they couldn't publish a TPC-C benchmark because SQLServer failed to handle "transparency" (a TPC benchmarking technical requirement).

The intended message for the audience, however, was that SQLServer is more than capable of exhibiting superior TPC-like benchmark performance, they just couldn't hack an official TPC number because of an annoying technicality. This is precisely the kind of self-promotional clandestine benchmarking that TPC was formed to discourage.

There is another problem in using a debit-credit workload. As the name indicates, debit-credit *implies* a simple ATM banking transaction like the TPC-A benchmark (now defunct) not the more complex transaction of TPC-C. A rule of thumb states that TPC-A throughput is about 5 times greater than TPC-C throughput[1]. One might guess that the Compaq/OLTP throughput would be more like 100,000 TPC-C equivalent transactions per minute. For historical reference, Tandem reported 20,000 tpmC over three years ago on a 120 node MIPS-R4000 Himalaya server.

Cluster (un)Availability

Attracted by the desire to support billions of desktop clients, Gray emphasized Microsoft's focus on cluster technologies. The essential idea is to strap multiple servers together using a high-performance interconnect network to enable both scalable performance and reliability (i.e., no single point of failure--see [3]). But the cluster concept is not new.

There are several historical precedents for scalable clusters that support commercial database workloads. In particular, Tandem for OLTP, Teradata for DSS (Decision Support Systems), and the IBM Parallel Sysplex. More recently Sun, HP, Sequent and others have developed and are developing UNIX cluster-based servers.

Despite Gray's claim that "NT clusters are easy," perhaps the most telling indicator of the current state of NT cluster technology was the failure to demonstrate 2-node failover! Another surprise: after the big wind-up, it was the small stuff that bombed!

Conclusion

I hope my review has given you some sense of my surprise. Be aware that I am NoT anti-NT by any means. On the contrary, from my standpoint NT looks like modern UNIX (commodity Mach?) with integrated windows--something I've wanted since my days at Xerox PARC. So, maybe it would be kinder and more accurate to retitle this opening piece: "NT to the Max...(NoT)...Yet."

References

[1] N.J. Gunther, *The Practical Performance Analyst*, McGraw-Hill, 1997. In press. See <http://members.aol.com/CoDynamo/Book.toc.htm> for publication status.

[2] Anon et al., "A Measure of Transaction Processing Power," *Datamation*,31,1 12, 1985.

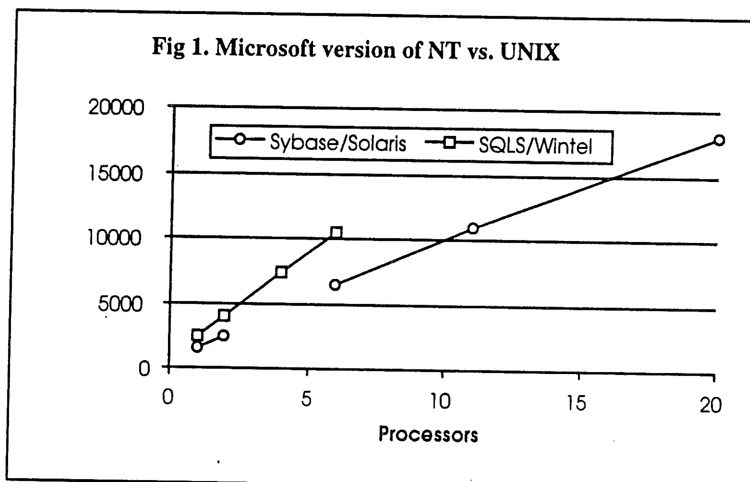
[3] The interested reader can learn more about this and the TPC-D Decision Support Systems (DSS) benchmark at <www.tpc.org>.

This article originally appeared in the USENIX Association journal ;`login:`, November 1997. We thank Dr. Gunther kindly for giving us permission to publish this article in AUUGN.

The ABC's of TPC's and NT Scalability II

Neil Gunther <ngunther@ricochet.net>
Copyright © 1998 Neil J. Gunther. All Rights Reserved.

In the last issue of ;login: [1] I promised to delve more into my concerns about the comparisons of UNIX and NT scalability that were presented at the USENIX-NT Workshop last August. In this second article I want to start with the data presented in Figure 1 which purported to show the superiority of NT over UNIX scalability on the common basis of the TPC-C benchmark workload.



Before doing so, however, I have to assume that most readers are not familiar with the TPC approach to database benchmarking. Unfortunately, there is not enough space to go into great detail about this complex measurement process, so I can only provide the briefest of sketches. The interested reader can find specifics at <www.tpc.org>.

TPC Road Rules

Unlike many computer benchmarks (e.g., Dhrystone, Linpack, SPEC) TPC benchmarks do not exist as code that you purchase or download. Rather, TPC provides a (downloadable) benchmarking specification document. Anyone, wishing to run the benchmark is free to implement the specification in any way they see fit. You are not free, however, to interpret the TPC rules as you please. In order to report an official TPC result you must write a corresponding full disclosure report that itemizes how you met each one of the clauses in the TPC specification. In addition, the benchmark runs that produced the result you wish to report must be witnessed and reviewed by an official TPC auditor at runtime. Your disclosure report is also reviewed by members of the TPC council. Any discrepancies that cannot be satisfactorily explained may lead to the result being withdrawn. In other words, TPC benchmarks are a serious and expensive undertaking that come with a high degree of credibility. Any attempt to cut corners is likely to be spotted and dealt with accordingly.

The TPC Performance Race

Currently, there are two TPC benchmarks: TPC-C (for benchmarking on-line database transaction processing; AKA OLTP systems) and TPC-D (for benchmarking decision support systems; AKA DSS). The TPC-A and TPC-B benchmarks have been retired for two major reasons:

- These workloads corresponded to a relatively simple debit/credit banking transaction.
- It stops ongoing attempts to exploit any loopholes in those older benchmark designs.

Moreover, both the TPC-A and TPC-B were directed solely at OLTP performance. TPC-C is a more complex OLTP benchmark that uses a heterogeneous mix of 5 transactions accessing a database that models inventory control in a distribute warehouse.

TPC-D is the first TPC benchmark to be directed at multi-user, large-scale, query-intensive systems. Rather than get bogged down in technical details, I've chosen to highlight the difference between TPC-C and TPC-D using the following whimsical analogy with automobile sporting events.

TPC-C Indianapolis 500
TPC-C is the Indy 500 of database benchmarking. In the real Indy event, 35 vehicles race around a 2.5 mile circuit and the first car over the finish line on the 200th lap is declared the winner. The sporting focus is on the performance of individual vehicles as measured by their top speeds in *miles per hour.*

In the TPC-C benchmark the database transactions are analogous to the Indy race-cars, but the performance focus is shifted away from the cars and onto the racetrack itself. For example, a wet track is slower than a dry one. The racetrack is analogous to the database platform and its performance could be measured by the number of *cars per minute* the track can support over the 200 loops of the Indy 500 race. It is a measure of the raceway's carrying capacity. Technically, this would be accomplished by counting the number of cars that cross the same place (e.g., the starting line) every 5 minutes (roughly the time it takes a car to make one loop of the raceway) and averaging those counts over the duration of the entire race. Under TPC road rules, any car taking longer than 5 minutes would not counted as part of the track's capacity. In the TPC version of the Indy 500, there is another rule that all cars must make at least one simultaneous pit-stop (corresponding to a database checkpoint) and then continue again.

In practice, when the checkered flag falls, all the cars take some time to maneuver into position and get up to top speed. In the TPC-C benchmark, this corresponds to the ramp-up period necessary to get the database cache warmed up and the system operating in steady-state. This ramp-up period is not included in the performance results. In the real TPC-C benchmark, transactions committed every half minute or so are counted and used to determine the average throughput measured as *transactions per minute* (or tpmC) over the entire benchmark run. Any transaction that does not commit within a 2 second response time is not counted.

That Transparency Thing
Furthermore, suppose you wanted to assess the Indy track capability on a worldwide basis
e.g., tracks in the USA, Australia, Canada, and Britain. This would be a way to compare Indy racing with other kinds of races e.g., NASCAR racing. The worldwide Indy performance would be given as the sum of the performance of each Indy raceway.

If you only raced USA cars on the USA track, Australian cars on the Australian track, and so on, you would be unintentionally optimizing the measurement. The TPC-C version of measuring this worldwide Indy performance does not permit such an optimization. Instead, you must also run some USA cars on the Australian raceway, and some Australian cars on the British track, and every other permutation in between. Moreover, which car runs on which track must be determined by drawing track-car pairs out of a hat. In other words, you are not allowed to bias the results by knowing beforehand which car will race on which track. The selection process is then said to be unbiased or *transparent*..

Similarly, in the real TPC-C benchmark you can have 4 servers with 4 separate database instances, but TPC-C does not permit you to confine transactions to each database separately and then add the separate throughputs together to give the total capacity. Transactions must be distributed in such a way that any transaction can access any of the 4 database tables without knowing ahead of time which database it will run against. This adds realism to the benchmark. But transparency can also introduce some performance degradation due to longer code paths needed to distribute the transactions.

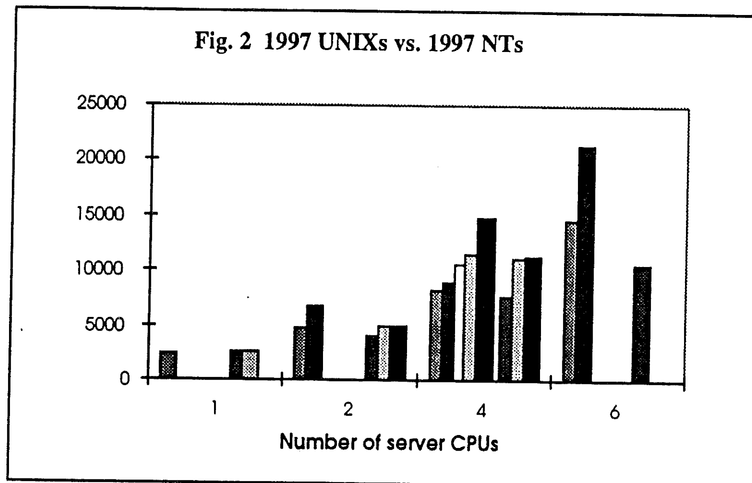
Clearly, it would be much simpler to ignore this transparency requirement and just add up the throughputs of more and more independent servers. That is an easy (but unrealistic) way to generate a big throughput number without any distribution overhead. So that's precisely what Microsoft did and, since it violates TPC-C road rules, they could not report it as a bona fide TPC-C result. It would never have got past the TPC auditor. Gray claimed this was just a "technicality"; now you can decide. On top of this failure, they didn't use TPC-C transactions either (contrary to the statement in [3]). What did they use? We'll never know because, not being a TPC benchmark, they were not subject to the disclosure rule. Gray used the term "debit-credit transaction" which suggests some kind of banking transaction, but we don't know that. I'd prefer to call it *diddleysquats per day* just to remind myself that the entire Microsoft claim is beFUDdled.

TPC-D Monster Tractor-Pull
In contrast to the TPC-C Indy 500 race, TPC-D is more like a monster tractor-pull. In the TPC-D version of the tractor-pull, there are 17 vehicles of different weights that the tractor must tow across the arena to complete the competition. For each tow, the elapsed time to get across the arena is measured and used to construct an overall towing capacity for the tractor. There's no constraint on how long it takes to pull all 17 vehicles since it's only the elapsed time for each pull that is measured. In the real TPC-D benchmark, the key performance metric is the time taken to execute each of the 17 queries. Gray did not discuss TPC-D results for SQLServer. Why not? Because there isn't any for SQLServer. You check for yourself at <http://www.tpc.org/execsum-_TPCD.html>. On the other hand, there many TPC-D results on UNIX.

Sensible Scalability Comparisons

Figure 2 shows a comparison of TPC-C results across a wide variety of results published in 1997. The most important notable difference from Figure 1 is that there are no curves. Why not? Because these are all different platforms running various flavors of UNIX, different RDBMSs, on different hardware. Using curves (as in Fig.1) would erroneously suggest that certain data belong to the same family, when they do not. Recall what I said about the performance analyst's cardinal rule [1]: Only change one thing at a time!

There are four CPU categories shown in Figure 2: uniprocessor, 2-way, 4-way, and 6-way multiprocessors. In CPU each category, the UNIX results are grouped together to the left while the NT results are grouped to the right. I've selected official TPC-C UNIX and NT results for all of 1997 to give some reasonable definition to my requirement that the data be in some sense contemporaneous [1]. The selected servers have between 1–6 processors to conform to the range where NT actually tries to compete with UNIX servers.



Things look a little less impressive for NT than in Gray's *benchmarketing* presentation [2]. First, note that there is considerable variance within each UNIX group. This is to be expected because (unlike NT) there is no single UNIX, and the data in Fig. 2 includes ORACLE and SYBASE running on various UNIX platforms. Typically, CPUs with larger second-level caches produce higher throughput because they can accommodate a large RDBMS footprint.

Second, there is far less variation within each NT group. This is to be expected when there is only one RDBMS (viz., SQLServer) tuned to tun a relatively few Intel-based architectures. Note also that for the 6-way configurations the best UNIX result (HP/SYBASE) has more than twice the throughput performance of the NT system, and the next best UNIX result (Sun/SYBASE) is more than 30% better than NT. This demolishes Gray's point based on Figure 1 that one needs to go to a more expensive 12-way UNIX system just to match a 6-way NT in throughput. How did I arrive at a different conclusion? I didn't bias the data by hand-picking aged Solaris/SYBASE TPC-C results for making comparisons.

Table 1 summarizes the various platform combinations that have been reported. (a) Sequent has announced a parallel query result on a 4 node NUMA-Q 2000 cluster with dual-quad CPUs (32 total CPUs). This is NOT a TPC-D result, however. Also, Compaq has an official TPC-C result with ORACLE on NT (third column in Fig.2). There are no SQLServer results on UNIX that I know of.

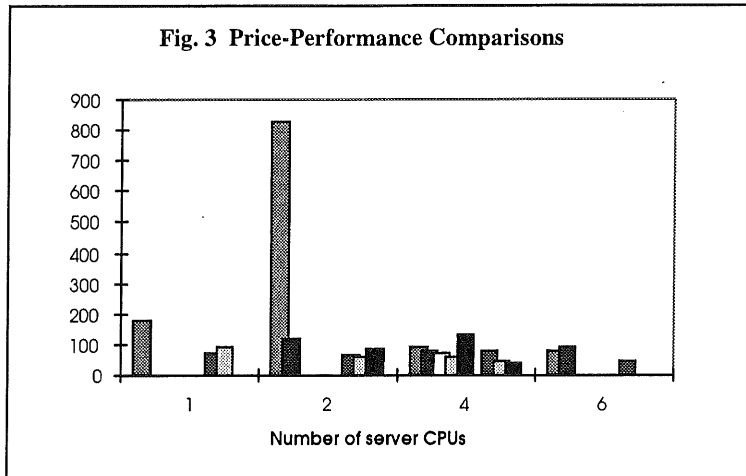
Table 1. Database and Platform combinations

RDBMS \ OS	NT	Solaris	UNIX
SQLServer	✔	✘	?
SYBASE	✔	✔	✔
Others	(a)	✔	✔

Price-Performance Comparisons

We can use the disclosed price of the TPC benchmark platform expressed as $/tpmC to make the comparisons shown in Fig. 3. When it comes to price-performance, Microsoft does indeed have the drop on UNIX; especially at the low end. But it's not so dramatic for larger CPU configurations. In case you're wondering, the expensive outlier in the 2-way class is a Fujitsu UNIX box.

Fig. 3 Price-Performance Comparisons



Since open system hardware is generally cheaper than mainframes, how is it that Wintel pricing beats UNIX so convincingly? One way of looking at this is simply, history repeating itself. Over the last 20 years UNIX workstations and multiprocessors have eroded the profit margins that were sacred to selling mainframe big-iron. This occurred because UNIX boxes were cheaper to build and became more ubiquitous than centralized mainframes. At the outset they could not compete with mainframe performance but gradually, that changed as UNIX systems scaled up.

Over the last 10 years, the PC has become more ubiquitous than UNIX servers. They represent real commodity computers. At the outset they could not compete with UNIX workstation or multiprocessor performance but gradually, that is changing as PC-based system scale up. In other words, the PC shall do unto UNIX servers what UNIX server hath done to mainframes.

Next time, I'll consider the factors that determine platform scalability.

Acknowledgments

I am grateful to Kim Shanley (TPC CEO), Francois Raab (TPC Auditor), and Mike Brey (Oracle) for various technical discussions.

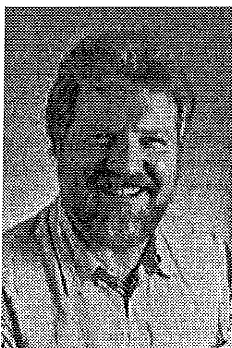
References

[1] N.J. Gunther, "NT to the Max...(NoT)," *;login:* pp.9-11, November 1997.

[2] J. Gray, "Windows NT to the Max," Original presentation slides are available at http://www.usenix.org/publications/library/proceedings/usenixnt97/presentations/index.html

[3] B. Dewey, Conference Report–Keynote Summary *;login:* pp.32-33, November 1997.

This article will appear in the USENIX Association journal ;login:, February 1998. We thank Dr. Gunther kindly for giving us permission to publish this article in AUUGN.



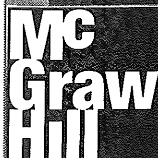
Neil Gunther is founder and principal consultant for Performance Dynamics Company™ in Mountain View, CA.

<http://members.aol.com/CoDynamo/Home.htm>

Dr. Gunther has worked in the Silicon Valley for 18 years. He is a member of IEEE, ACM, and CMG.

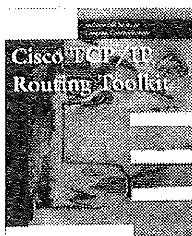
New

Professional Computing Titles



FROM McGRAW-HILL

Cisco TCP/IP Routing Professional Toolkit



Chris Lewis

0.07.041088.7

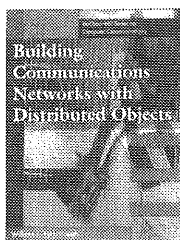
softcover

$110.00

AUUG $88.00

With the proliferation of TCP/IP networks and Cisco's vast share of the router market, this toolkit is an essential for MIS directors, LAN managers and administrators, network engineers and technical support staff. The book explains: Setting up your first router; System implementation; Protocol basics; Performance troubleshooting; Network management; Security. This comprehensive guide also offers solutions to everyday problems inherent in network use (eg. recovery of lost passwords), plus an overview of Cisco products.

Building Communications Networks with Distributed Objects



William Yarborough

0.07.072220.X

softcover

$110.00

AUUG $88.00

This guide is a powerful ally for the computing professional working in the client/server arena. The book is not only a thorough primer on using distributed object technology to integrate networks and to support client/server applications, it also offers comprehensive treatments of middleware, vital security considerations, and every issue of concern to those charged with building today's crucial corporate networks.

Firewalls Complete

Marcus Goncalves

0.07.024645.9

softcover+ CD ROM

$115.00

AUUG $92.00

What's the best way to ensure Internet security? With firewalls. And the best way to learn firewall installation and maintenance from A to Z? This info-packed guide covers virtually all firewall techniques, technologies, and brands- and even includes a blueprint for designing your own. The CD ROM contains 20 firewall product demos and evaluations from such major vendors as Check Point, IBM and Sun.

Qty	ISBN		Title	Members price (20% OFF)	ARRP $
	0.07.041088.7	Lewis	Cisco TCP/IP Routing Toolkit	$88.00	$110.00
	0.07.072220.X	Yaraborough	Building Communications Networks with Distributed Objects	$88.00	$110.00
	COMPLETE SERIES				
	0.07.024645.9	Gonclaves	Firewalls Complete	$92.00	$115.00
			SUBTOTAL		
			POSTAGE & HANDLING	$6.00	
			TOTAL		

FAX YOUR ORDER NOW ON (02) 9417 7003 OR PHONE: (02) 9415 9888

I enclose payment for the above book(s) by ☐ *cheque* ☐ *credit card* ☐ *company purchase order*

Personal Details:

Name ____________________

Address ____________________

________ State ______ Postcode ________

Phone ____________ Fax ____________

Please charge my:

☐ Bankcard ☐ Mastercard ☐ Visa ☐ Diners Club ☐ Amex

No. ____________________

Signed ____________________

Expiry Date ____________________

McGraw-Hill Book Company Australia Pty Ltd

4 Barcoo Street, Roseville NSW 2069, Phone: (02) 9415 9888 Fax: (02) 9417 7003. E-mail: profref@mcgraw-hill.com.au

AUUG298

THE PRACTICAL PERFORMANCE ANALYST

by Dr. Neil Gunther

(published by McGraw-Hill)

Following Neil's successful tour around Australia, in which he presented, **"High Performance WEB and DATABASE Techniques"**, we are pleased to offer a copy of his latest book to AUUG members at the specially discounted price of **$90 per copy** (rrp $160).

Please complete the McGraw-Hill order form on the facing page to ensure you don't miss out!

The book covers the following topics:

PART I - FOUNDATIONS

Chapter 1 About Time!
Chapter 2 Queuing theory for those who can't wait
Chapter 3 Systems of queues
Chapter 4 Distributed performance management

PART II - APPLICATIONS

Chapter 5 Commercial parallelism
Chapter 6 Parallel systems
Chapter 7 Multiprocessor systems
Chapter 8 Client/Server applications
Chapter 9 Web servers

PART III - INNOVATIONS

Chapter 10 Small numbers, BIG consequences
Chapter 11 Paths, potentials, and probabilities
Chapter 12 Large transients in packet-switched networks
Chapter 13 Large transients in circuit-switched networks
Chapter 14 The dynamics of scaling

PART IV - APPENDICES

Appendix A PDQ© (Pretty Damn Quick) user manual
Appendix B Performance organizations
Appendix C Guidelines for Making Multiprocessor Applications Symmetric
(Contributed by Robert M. Lane)
Appendix D Glossary of terms
Bibliography

Recursive Make Considered Harmful

Peter Miller <millerp@canb.auug.org.au>
Copyright (C) 1997 Peter Miller. All rights reserved.

Abstract
For large UNIX projects, the traditional method of building the project is to use recursive *make.* On some projects, this results in build times which are unacceptably large, when all you want to do is change one file. In examining the source of the overly long build times, it became evident that a number of apparently unrelated problems combine to produce the delay, but on analysis all have the same root cause.

This paper explores an number of problems regarding the use of recursive make, and shows that they are all symptoms of the same problem. Symptoms that the UNIX community have long accepted as a fact of life, but which need not be endured any longer. These problems include recursive *makes* which take ``forever" to work out that they need to do nothing, recursive *makes* which do too much, or too little, recursive *makes* which are overly sensitive to changes in the source code and require constant `Makefile` intervention to keep them working.

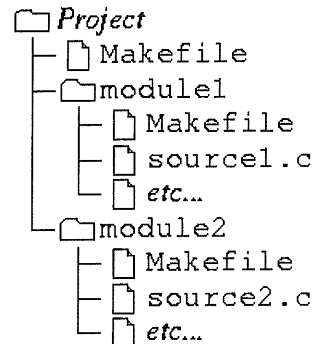
The resolution of these problems can be found by looking at what *make* does, from first principles, and then analyzing the effects of introducing recursive *make* to this activity. The analysis shows that the problem stems from the artificial partitioning of the build into separate subsets. This, in turn, leads to the symptoms described. To avoid the symptoms, it is only necessary to avoid the separation; to use a single `Makefile` for the whole project.

This conclusion runs counter to much accumulated folk wisdom in building large projects on UNIX. Some of the main objections raised by this folk wisdom are examined and shown to be unfounded. The results of actual use are far more encouraging, with routine development performance improvements significantly faster than intuition may indicate. The use of a single project `Makefile` is not as difficult to put into practice as it may first appear.

1. Introduction
For large UNIX software development projects, the traditional methods of building the project use what has come to be known as ``recursive *make.*" This refers to the use of a hierarchy of directories containing source files for the modules which make up the project, where each of the sub-directories contains a *Makefile* which describes the rules and instructions for the *make* program. The complete project build is done by arranging for the top-level `Makefile` to change directory into each of the sub-directories and recursively invoke *make.*

This paper explores some significant problems encountered when developing software projects using the recursive *make* technique. A simple solution is offered, and some of the implications of that solution are explored.

Recursive *make* results in a directory tree which looks something like this:



This hierarchy of modules can be nested arbitrarily deep. Real-world projects often use two- and three-level structures.

1.1 Assumed Knowledge
This paper assumes that the reader is familiar with developing software on UNIX, with the *make* program, and with the issues of C programming and include file dependencies.

This paper assumes that you have installed GNU Make on your system and are moderately familiar with its features. Some features of *make* described below may not be available if you are using the limited version supplied by your vendor.

2. The Problem
There are numerous problems with recursive *make,* and they are usually observed daily in practice. Some of these problems include:

- It is very hard to get the *order* of the recursion into the sub-directories correct. This order is very unstable and frequently needs to be manually ``tweaked." Increasing the number of directories, or increasing the depth in the directory tree, cause this order to be increasingly unstable.
- It is often necessary to do more than one pass over the sub-directories to build the whole system. This, naturally, leads to extended build times.
- Because the builds take so long, some dependency information is omitted, otherwise development builds take unreasonable lengths of

time, and the developers are unproductive. This usually leads to things not being updated when they need to be, requiring frequent ``clean'' builds from scratch, to ensure everything has actually been built.

- Because inter-directory dependencies are either omitted or too hard to express, the `Makefiles` are often written to build *too much* to ensure that nothing is left out.
- The inaccuracy of the dependencies, or the simple lack of dependencies, can result in a product which is incapable of building cleanly, requiring the build process to be carefully watched by a human.

3. Analysis

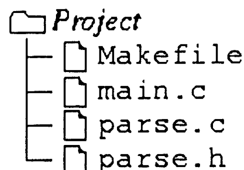
Before it is possible to address these seemingly unrelated problems, it is first necessary to understand what *make* does and how it does it. It is then possible to look at the effects recursive *make* has on how *make* behaves.

3.1. Whole Make

Make is an expert system. You give it a set of rules for how to construct things, and a target to be constructed. The rules can be decomposed into pair-wise ordered dependencies between files. *Make* takes the rules and determines how to build the given target. Once it has determined how to construct the target, it proceeds to do so.

Make determines how to build the target by constructing a *directed acyclic graph*, the DAG familiar to many Computer Science students. The vertices of this graph are the files in the system, the edges of this graph are the inter-file dependencies. The edges of the graph are directed because the pair-wise dependencies are ordered; resulting in a *acyclic* graph – things which look like loops are resolved by the direction of the edges.

This paper will use a small example project for its analysis. While the number of files in this example is small, there is sufficient complexity to demonstrate all of the above recursive *make* problems. First, however, the project is presented in a non-recursive form.

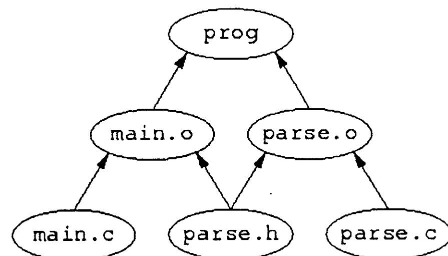


The `Makefile` in this small project looks like this:

```
OBJ = main.o parse.o

prog: $(OBJ)
   $(CC) -o $@ $(OBJ)

main.o: main.c parse.h
   $(CC) -c main.c

parse.o: parse.c parse.h
   $(CC) -c parse.c
```

Some of the implicit rules of *make* are presented here explicitly, to assist the reader in converting the `Makefile` into its equivalent DAG.

The above `Makefile` can be drawn as a DAG in the following form:



This is an *acyclic* graph is because of the arrows which express the ordering of the relationship between the files. If there *was* a circular dependency according to the arrows, it would be an error.

Note that the object files (`.o`) are dependent on the include files (`.h`) even though it is the source files (`.c`) which do the including. This is because if an include file changes, it is the object files which are out-of-date, not the source files.

The second part of what *make* does it to perform a *postorder* traversal of the DAG. That is, the dependencies are visited first. The actual order of traversal is undefined, but most *make* implementations work down the graph from left to right for edges below the same vertex, and most projects implicitly rely on this behavior. The last-time-modified of each file is examined, and higher files are determined to be out-of-date if any of the lower files on which they depend are younger. Where a file is determined to be out-of-date, the action associated with the relevant graph edge is performed (in the above example, a compile or a link).

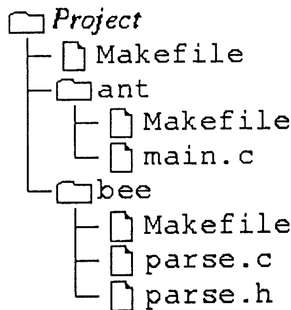
The use of recursive *make* affects both phases of the operation of *make*: it causes *make* to construct an inaccurate DAG, and it forces *make* to traverse the DAG in an inappropriate order.

3.2. Recursive Make

To examine the effects of recursive *makes*, the above example will be artificially segmented into two modules, each with its own `Makefile`, and a top-

level `Makefile` used to invoke each of the module `Makefiles`.

The directory structure is as follows:



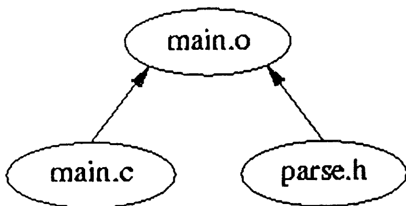
The top-level `Makefile` looks like this:

```
MODULES = ant bee

all:
   for dir in $(MODULES); do \
      (cd $$dir; ${MAKE} all); \
   done
```

The `ant/Makefile` looks like this:

```
all: main.o

main.o: main.c ../bee/parse.h
   $(CC) -I../bee -c main.c
```

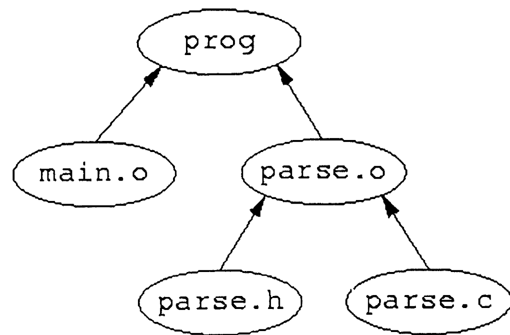
and the equivalent DAG looks like this:



The `bee/Makefile` looks like this:

```
OBJ = ../ant/main.o parse.o

all: prog

prog: (OBJ)
$(CC) -o $@ $(OBJ)

parse.o: parse.c parse.h
$(CC) -c parse.c
```

and the equivalent DAG looks like this:

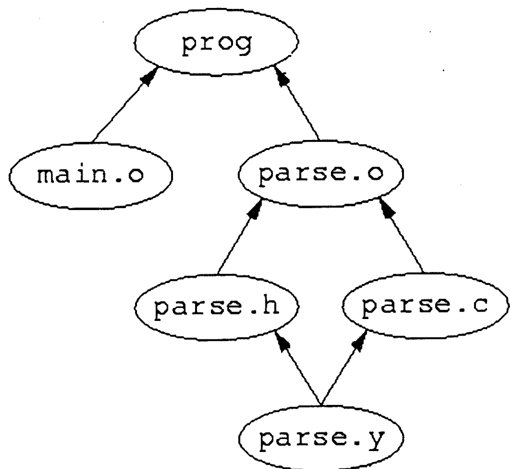


Take a close look at the DAGs. Notice how neither is complete - there are vertices and edges (files and dependencies) missing from both DAGs. When the entire build is done from the top level, everything will work.

But what happens when small changes occur? For example, what would happen of the `parse.c` and `parse.h` files were generated from a `parse.y` yacc grammar? This would add the following lines to the `bee/Makefile`:

```
parse.c parse.h: parse.y
   $(YACC) -d parse.y
   mv y.tab.c parse.c
   mv y.tab.h parse.h
```

And the equivalent DAG changes to look like this:



This change has a simple effect: if `parse.y` is edited, `main.o` will **not** be constructed correctly. The is because the DAG for `ant` knows about only some of the dependencies of `main.o`, and the DAG for `bee` knows none of them.

To understand why this happens, it is necessary to look at the actions *make* will take *from the top level*. Assume that the project is in a self-consistent state. Now edit `parse.y` in such a way that the generated `parse.h` file will have non-trivial differences. However, when the top-level *make* is invoked, first

ant and then bee is visited. But ant/main.o is *not* recompiled, because bee/parse.h has not yet been regenerated and thus does not yet indicate that main.o is out-of-date. It is not until bee is visited by the recursive *make* that parse.c and parse.h are reconstructed, followed by parse.o. When the program is linked main.o and parse.o are non-trivially incompatible. That is, the program is *wrong*.

3.3. Traditional Solutions

There are three traditional fixes for the above ``glitch.''

3.3.1. Reshuffle

The first is to manually tweak the order of the modules in the top-level Makefile. But why is this tweak required at all? Isn't *make* supposed to be an expert system? Is *make* somehow flawed, or did something else go wrong?

To answer this question, it is necessary to look, not at the graphs, but the *order of traversal* of the graphs. In order to operate correctly, make needs to perform a *postorder* traversal, but in separating the DAG into two pieces, *make* has not been *allowed* to traverse the graph in the necessary order - instead the project has dictated an order of traversal. An order which, when you consider the original graph, is plain *wrong*. Tweaking the top-level Makefile corrects the order to one similar to that which *make* could have used. Until the next dependency is added...

3.3.2. Repetition

The second traditional solution is to make more than one pass in the top-level Makefile, something like this:

```
MODULES = ant bee

all:
   for dir in $(MODULES); do \
      (cd $$dir; ${MAKE} all); \
   done
   for dir in $(MODULES); do \
      (cd $$dir; ${MAKE} all); \
   done
```

This doubles then length of time it takes to perform the build. But that is not all: there is no guarantee that two passes are enough! The upper bound of the number of passes is not even proportional to the number of modules, it is instead proportional to the number of graph edges which cross module boundaries.

3.3.3. Overkill

We have already seen an example of how recursive *make* can build too little, but another common problem is to build too much. The third traditional solution to the above glitch is to add even *more* lines to ant/Makefile:

```
.PHONY: ../bee/parse.h

../bee/parse.h:
  cd ../bee; \
  make clean; \
  make all
```

This means that whenever main.o is made, parse.h will always be considered to be out-of-date. All of bee will always be rebuilt including parse.h, and so main.o will always be rebuilt, *even if everything was self consistent.*

4. Prevention

The above analysis is based one simple action: the DAG was artificially separated into incomplete pieces. This separation resulted in all of the problems familiar to recursive *make* builds.

Did make get it wrong? No. This is a case of the ancient GIGO principle: Garbage In, Garbage Out. Incomplete Makefiles are wrong Makefiles.

To avoid these problems, don't break the DAG into pieces; instead, use one Makefile for the entire project. It is not the recursion itself which is harmful, it is the crippled Makefiles which are used in the recursion which are wrong. It is not a deficiency of make itself that recursive make is broken, it does the best it can with the flawed input it is given.

> ``*But, but, but... You can't do that!*'' I hear you cry. ``*A single* Makefile *is too big, it's unmaintainable, it's too hard to write the rules, you'll run out of memory, I only want to build my little bit, the build take too long. It's just not practical.*''

These are valid concerns, and they frequently lead make users to the conclusion that re-working their build process does not have any short- or long-term benefits. This conclusion is based on ancient, enduring, false assumptions.

4.1. A Single Makefile Is Too Big

If the entire project build description were placed into a single Makefile this would certainly be true, however modern make implementations have *include* statements. By including a relevant fragment from each module, the total size of the Makefile and its include files need be no larger than the total size of the Makefiles in the recursive case.

4.2. A Single Makefile Is Unmaintainable

The complexity of using a single top-level Makefile which includes a fragment from each module is no more complex than in the recursive case. Because the DAG is not segmented, this form of Makefile becomes less complex, and thus *more* maintainable, simply because fewer ``tweaks'' are required to keep it working.

Recursive Makefiles have a great deal of repetition. Many projects solve this by using include files. By using a single Makefile for the project, the need for the ``common" include files disappears - the single Makefile is the common part.

4.3. It's Too Hard To Write The Rules

The only change required is to include the directory part in filenames in a number of places. This is because the *make* is performed from the top-level directory; the current directory is not the one in which the file appears. Where the output file is explicitly stated in a rule, this is not a problem.

GCC allows a -o option in conjunction with the -c option, and GNU Make knows this. This results in the implicit compilation rule placing the output in the correct place. Older and dumber C compilers, however, may not allow the -o option with the -c option, and will leave the object file in the top-level directory (*i.e.* the wrong directory). There are three ways for you to fix this: get GNU Make and GCC, override the built-in rule with one which does the right thing, or complain to your vendor.

Also, K&R C compilers will start the double-quote include path (#include *"filename.h"*) from the current directory. This will not do what you want. ANSI C compliant C compilers, however, start the double-quote include path from the directory in which the source file appears; thus, no source changes are required. If you don't have an ANSI C compliant C compiler, you should consider installing GCC on your system as soon as possible.

4.4. I Only Want To Build My Little Bit

Most of the time, developers are deep within the project tree and they edit one or two files and then run make to compile their changes and try them out. They may do this dozens or hundreds of times a day. Being forced to do a full project build every time would be absurd.

Developers always have the option of giving make a specific target. This is always the case, it's just that we usually rely on the default target in the Makefile in the current directory to shorten the command line for us. Building ``my little bit" can still be done with a whole project Makefile, simply by using a specific target, and an alias if the command line is too long.

Is doing a full project build every time so absurd? If a change made in a module has repercussions in other modules, because there is a dependency the developer is unaware of (but the Makefile is aware of), isn't it better that the developer find out as early as possible? Dependencies like this will be found, because the DAG is more complete than in the recursive case.

The developer is rarely a seasoned old salt who knows every one of the million lines of code in the product. More likely the developer is a short-term contractor or a junior. You don't want implications like these to blow up after the changes are integrated with the master source, you want them to blow up on the developer in some nice safe sand-box far away from the master source.

If you want to make ``just your little" bit because you are concerned that performing a full project build will corrupt the project master source, due to the directory structure used in your project, see the ``Projects *versus* Sand-Boxes" section below.

4.5. The Build Will Take Too Long

This statement can be made from one of two perspectives. First, that a whole project *make*, even when everything is up-to-date, inevitably takes a long time to perform. Secondly, that these extended delays are unacceptable when a developer wants to quickly compile and link the one file that they have changed.

4.5.1. Project Builds

Consider a hypothetical project with 1000 source (.c) files, each of which has its calling interface defined in a corresponding include (.h) file with defines, type declarations and function prototypes. These 1000 source files include their own interface definition, plus the interface definitions of any other module they may call. These 1000 source files are compiled into 1000 object files which are then linked into an executable program. This system has some 3000 files which *make* must be told about, and be told about the include dependencies, and also explore the possibility that implicit rules (.y → .c for example) may be necessary.

In order to build the DAG, *make* must ``stat" 3000 files, plus an additional 2000 files or so, depending on which implicit rules your make knows about and your Makefile has left enabled. On the author's humble 66MHz i486 this takes about 10 seconds; on native disk on faster platforms it goes even faster. With NFS over 10MB Ethernet it takes about 10 seconds, no matter what the platform.

This is an astonishing statistic! Imagine being able to a single file compile, out of 1000 source files, in only 10 seconds, plus the time for the compilation itself.

Breaking the set of files up into 100 modules, and running it as a recursive *make* takes about 25 seconds. The repeated process creation for the subordinate *make* invocations take quite a long time.

Hang on a minute! On real-world projects with less than 1000 files, it takes an awful lot longer than 25 seconds for *make* to work out that it has nothing to do. For some projects, doing it in only 25 minutes would be an improvement! The above result tells us that it is not the number of files which is slowing us down (that only takes 10 seconds), and it is not the repeated process creation for the subordinate *make*

invocations (that only takes another 15 seconds). So just what is taking so long?

The traditional solutions to the problems introduced by recursive make often increase the number of subordinate make invocations beyond the minimum described here; e.g. to perform multiple repetitions (3.3.2), or to overkill cross-module dependencies (3.3.3). These can take a long time, particularly when combined, but do not account for some of the more spectacular build times; what else is taking so long?

Complexity of the `Makefile` is what is taking so long. This is covered, below, in the *Efficient* `Makefiles` section.

4.5.2. Development Builds

If, as in the 1000 file example, it only takes 10 seconds to figure out which one of the files needs to be recompiled, there is no serious threat to the productivity of developers if they do a whole-project *make* as opposed to a module-specific *make*. The advantage for the project is that the module-centric developer is reminded at relevant times (and only relevant times) that their work has wider ramifications.

By consistently using C include files which contain accurate interface definitions (including function prototypes), this will produce compilation errors in many of the cases which would result in a defective product. By doing whole-project builds, developers discover such errors very early in the development process, and can fix the problems when they are least expensive.

4.6. You'll Run Out Of Memory

This is the most interesting response. Once long ago, on a CPU far, far away, it may even have been true. When Feldman [1] first wrote *make* it was 1978 and he was using a PDP11. Unix processes were limited to 64KB of data.

On such a computer, the above project with its 3000 files detailed in the whole-project `Makefile`, would probably *not* allow the DAG and rule actions to fit in memory.

But we are not using PDP11s any more. The physical memory of modern computers exceeds 10MB for *small* computers, and virtual memory often exceeds 100MB. It is going to take a project with hundreds of thousands of source files to exhaust virtual memory on a *small* modern computer. As the 1000 source file example takes less than 100KB of memory (try it, I did) it is unlikely that any project manageable in a single directory tree on a single disk will exhaust your computer's memory.

4.7. Why Not Fix The DAG In The Modules?

It was shown in the above discussion that the problem with recursive *make* is that the DAGs are incomplete. It follows that by adding the missing portions, the problems would be resolved without abandoning the existing recursive *make* investment.

- The developer needs to remember to do this. The problems will not affect the developer of the module, it will affect the developers of *other* modules. There is no trigger to remind the developer to do this, other than the ire of fellow developers.
- It is difficult to work out where the changes need to be made. Potentially every `Makefile` in the entire project needs to be examined for possible modifications. Of course, you can wait for your fellow developers to find them for you.
- The include dependencies will be recomputed unnecessarily, or will be interpreted incorrectly. This is because *make* is string based, and thus ``." and ``../ant" are two different places, even when you are in the ant directory. This is of concern when include dependencies are automatically generated - as they are for all large projects.

By making sure that each `Makefile` is complete, you arrive at the point where the `Makefile` for at least one module contains the equivalent of a whole-project `Makefile` (recall that these modules form a single project and are thus inter-connected), and there is no need for the recursion any more.

5. Efficient `Makefiles`

The central theme of this paper is the *semantic* side-effects of artificially separating a `Makefile` into the pieces necessary to perform a recursive *make*. However, once you have a large number of `Makefiles`, the speed at which *make* can interpret this multitude of files also becomes an issue.

Builds can take ``forever" for both these reasons: the traditional fixes for the separated DAG may be building too much *and* your `Makefile` may be inefficient.

5.1. Deferred Evaluation

The text in a `Makefile` must somehow be read from a text file and understood by *make* so that the DAG can be constructed, and the specified actions attached to the edges. This is all kept in memory.

The input language for `Makefiles` is deceptively simple. A crucial distinction that often escapes both novices and experts alike is that *make's* input language is *text based*, as opposed to token based, as is the case for C or even the Unix shell. *Make* does the very least possible to process input lines and stash them away in memory.

As an example of this, consider the following assignment:

```
OBJ = main.o parse.o
```

Humans read this as the variable OBJ being assigned two filenames ``main.o'' and ``parse.o''. But *make* does not see it that way. Instead OBJ is assigned the *string* ``main.o parse.o''. It gets worse:

```
SRC = main.c parse.c
OBJ = $(SRC:.c=.o)
```

In this case humans expect *make* to assign the same two filenames to OBJ, but *make* actually assigns the string ``$(SRC:.c=.o)''. This is because it is a *macro* language with deferred evaluation, as opposed to one with variables and immediate evaluation.

If this does not seem too problematic, consider the following Makefile:

```
SRC = $(shell echo 'Ouch!' \
   1>&2 ; echo *.[cy])
OBJ = \
   $(patsubst %.c,%.o,\
      $(filter %.c,$(SRC))) \
   $(patsubst %.y,%.o,\
      $(filter %.y,$(SRC)))

test: $(OBJ)
   $(CC) -o $@ $(OBJ)
```

How many times will the shell command be executed? **Ouch!** It will be executed *twice* just to construct the DAG, and a further *two* times if the rule needs to be executed.

If this shell command does anything complex or time consuming (and it usually does) it will take *four* times longer than you thought.

But it is worth looking at the other portions of that OBJ macro. Each time it is named, a huge amount of processing is performed:

- The argument to *shell* is a single string (all built-in-functions take a single string argument). The string is executed in a sub-shell, and the standard output of this command is read back in, translating newlines into spaces. The result is a single string.
- The argument to *filter* is a single string. This argument is broken into two strings at the first comma. These two strings are then each broken into sub-strings separated by spaces. The first set are the patterns, the second set are the filenames. Then, for each of the pattern sub-strings, if a filename sub-string matches it, that filename is included in the output. Once all of the output has been found, it is re-assembled into a single space-separated string.
- The argument to *patsubst* is a single string. This argument is broken into three strings at the first and second commas. The third string is then broken into sub-strings separated by spaces, these are the filenames. Then, for each of the filenames which match the first string it is substituted according to the second string. If a filename does not match, it is passed through unchanged. Once all of the output has been generated, it is re-assembled into a single space-separated string.

Notice how many times those strings are disassembled and re-assembled. Notice how many ways that happens. *This is slow.* The example here names just two files but consider how inefficient this would be for 1000 files. Doing it *four* times becomes decidedly inefficient.

If you are using a dumb *make* that has no substitutions and no built-in functions, this cannot bite you. But a modern *make* has lots of built-in functions and can even invoke shell commands on-the-fly. The semantics of *make's* text manipulation is such that string manipulation in *make* is very CPU intensive, compared to performing the same string manipulations in C or AWK.

5.2. Immediate Evaluation

Modern *make* implementations have an immediate evaluation ``:='' assignment operator. The above example can be re-written as:

```
SRC := $(shell echo 'Ouch!' \
   1>&2 ; echo *.[cy])
OBJ := \
   $(patsubst %.c,%.o,\
      $(filter %.c,$(SRC))) \
   $(patsubst %.y,%.o,\
      $(filter %.y,$(SRC)))

test: $(OBJ)
   $(CC) -o $@ $(OBJ)
```

Note that *both* assignments are immediate evaluation assignments. If the first were not, the shell command would always be executed twice. If the second were not, the expensive substitutions would be performed at least twice and possibly four times.

As a rule of thumb: always use immediate evaluation assignment unless you knowingly want deferred evaluation.

5.3. Include Files

Many Makefiles perform the same text processing (the filters above, for example) for every single *make* run, but the results of the processing rarely change. Wherever practical, it is more efficient to record the results of the text processing into a file, and have the Makefile include this file.

5.4. Dependencies

Try to be careful about include files. They are relatively inexpensive to read, so more rather than less doesn't greatly affect efficiency.

As an example of this, it is first necessary to describe a useful feature of GNU Make: once a `Makefile` has been read in, if any of its included files were out-of-date (or do not yet exist), they are re-built, and then *make* starts again, which has the result that *make* is now working with up-to-date include files. This feature can be exploited to obtain automatic include file dependency tracking for C sources. The obvious way to implement it, however, has a subtle flaw.

```
SRC := $(wildcard *.c)
OBJ := $(SRC:.c=.o)

test: $(OBJ)
   $(CC) -o $@ $(OBJ)

include dependencies

dependencies: $(SRC)
   depend.sh $(CFLAGS) \
     $(SRC) > $@
```

The depend.sh script prints lines of the form:

file.o: file.c include.h ...

The most simple implementation of this is to use GCC, but you will need to edit the output if you want to avoid dependencies on system include files:

```
#!/bin/sh
gcc -M -MG $* |
sed 's@ /[^ ]*@@g'
```

This implementation of tracking C include dependencies has several serious flaws, but the one most commonly discovered is that the `dependencies` file does not, itself, depend on the C include files. That is, it is not re-built in one of the include files changes. There is no edge in the DAG joining the `dependencies` vertex to any of the include file vertices. If an include file changes to include another file (a nested include), the dependencies will not be recalculated, and potentially the C file will not be recompiled, and thus the program will not be re-built correctly.

A classic build-too-little problem, caused by giving make inadequate information, and thus causing it to build an inadequate DAG and reach the wrong conclusion.

The traditional solution is to build too much:

```
SRC := $(wildcard *.c)
OBJ := $(SRC:.c=.o)

test: $(OBJ)
   $(CC) -o $@ $(OBJ)

include dependencies

.PHONY: dependencies

dependencies: $(SRC)
   depend.sh $(CFLAGS) \
     $(SRC) > $@
```

Now, even if the project is completely up-do-date, the dependencies will be re-built. For a large project, this is very wasteful, and can be a major contributor to *make* taking ``forever'' to work out than nothing needs to be done.

There is a second problem, and that is that if any one of the C files changes, *all* of the C files will be re-scanned for include dependencies. This is as inefficient as having a `Makefile` which reads:

```
prog: $(SRC)
   $(CC) -o $@ $(SRC)
```

What is needed, in exact analogy to the C case, is to have an intermediate form. This is usually given a ``.d'' suffix. By exploiting the fact that more than one file may be named in an include line, there is no need to ``link'' all of the ``.d'' files together:

```
SRC := $(wildcard *.c)
OBJ := $(SRC:.c=.o)

test: $(OBJ)
   $(CC) -o $@ $(OBJ)

include $(OBJ:.o=.d)

%.d: %.c
   depend.sh $(CFLAGS) $< > $@
```

This has one more thing to fix: just as the object (`.o`) files depend on the source files and the include files, so do the dependency (`.d`) files. This means tinkering with the `depend.sh` script once more:

```
#!/bin/sh
gcc -M -MG $* |
sed -e 's@ /[^ ]*@@g' \
  -e 's@^\(.*\)\.o:@\1.d \1.o:@'
```

This method of determining include file dependencies results in the `Makefile` including more files than the original method, but opening files is less expensive than rebuilding all of the

dependencies every time. Typically a developer will edit one or two files before re-building; this method will rebuild the *exact* dependency file affected (or more than one, if you edited an include file). On balance, this will use less CPU, and less time.

In the case of a build where nothing needs to be done, *make* will actually do nothing, and would work this out very quickly.

5.5. Multiplier

All of the inefficiencies described in this section compound together. If you do 100 `Makefile` interpretations, once for each module, checking 1000 source files can take a very long time - if the interpretation requires complex processing or performs unnecessary work, or both. A whole project *make*, on the other hand, only needs to interpret a single `Makefile`.

6. Projects *versus* Sand-boxes

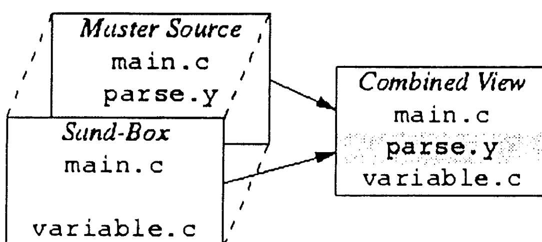
The above discussion assumes that a project resides under a single directory tree, and this is often the ideal. However, the realities of working with large software projects often lead to weird and wonderful directory structures in order to have developers working on different sections of the project without taking complete copies and thereby wasting precious disk space.

It is possible to see the whole-project *make* proposed here as impractical, because it does not match the evolved methods of your development process.

The whole-project *make* proposed here does have an effect on development methods: it can give you cleaner and simpler build environments for your developers. By using *make's* `VPATH` feature, it is possible to copy only those files you need to edit into your private work area, often called a *sand-box.*

The simplest explanation of what `VPATH` does is to make an analogy with the include file search path specified using `-I`*path* options to the C compiler. This set of options describes where to look for files, just as `VPATH` tells *make* where to look for files.

By using `VPATH`, it is possible to ``stack" the sand-box *on top of* the project master source, so that files in the sand-box take precedence, but it is the union of all the files which *make* uses to perform the build.



In this environment, the sand-box has the same tree structure as the project master source. This allows developers to safely change things across separate modules, *e.g.* if they are changing a module interface. It also allows the sand-box to be physically separate - perhaps on a different disk, or under their home directory. It also allows the project master source to be read-only, if you have (or would like) a rigorous check-in procedure.

Note: in addition to adding a `VPATH` line to your development `Makefile`, you will also need to add `-I` options to the `CFLAGS` macro, so that the C compiler uses the same path as *make* does. This is simply done with a 3-line `Makefile` in your work area - set a macro, set the `VPATH`, and then include the `Makefile` from the project master source.

6.1. Patch

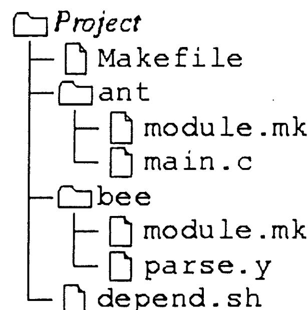
The POSIX semantics of `VPATH` are slightly brain-dead, and GNU Make is POSIX compliant, so the above discussion assumes you have a GNU Make with Paul Smith's `VPATH+` patch applied. This may be obtained from `ftp://ftp.wellfleet.-com/netman/psmith/gmake/`. Here is an extract from the README file:

> Once GNU make finds a target file through `VPATH/vpath` it changes the target path to the `VPATH/vpath` name immediately, causing all dependencies of the target to be searched for in the `VPATH` directory to the exclusion of local files which might be newer.

See the actual README file for a longer explanation.

7. The Big Picture

This section brings together all of the preceding discussion, and presents the example project with its separate modules, but with a whole-project `Makefile`. The directory structure is changed little from the recursive case, except that the deeper `Makefiles` are replaced by module specific include files:



The `Makefile` looks like this:

```
MODULES := ant bee

# look for include files in
#   each of the modules
CFLAGS += $(patsubst %,-I%,\
   $(MODULES))

# extra libraries if required
LIBS :=

# each module will add to this
SRC :=

# include the description for
#   each module
include $(patsubst %,\
   %/module.mk,$(MODULES))

# determine the object files
OBJ := \
   $(patsubst %.c,%.o,\
      $(filter %.c,$(SRC))) \
   $(patsubst %.y,%.o,\
      $(filter %.y,$(SRC)))

# link the program
prog: $(OBJ)
   $(CC) -o $@ $(OBJ) $(LIBS)

# include the C include
#   dependencies
include $(OBJ:.o=.d)

# calculate C include
#   dependencies
%.d: %.c
   depend.sh $(CFLAGS) $< > $@
```

The `ant/module.mk` file looks like:

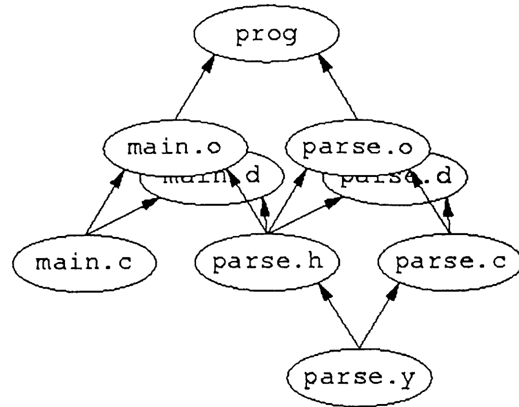
```
SRC += ant/main.c
```

The `bee/module.mk` file looks like:

```
SRC += bee/parse.y
LIBS += -ly

%.c %.h: %.y
   $(YACC) -d %.y
   mv y.tab.c %.c
   mv y.tab.h %.h
```

Notice that the built-in rules are used for the C files, but we need special yacc processing to get the generated `.h` file.

The equivalent DAG of the `Makefile` after all of the includes looks like this:



The vertexes and edges for the include file dependency files are also present as these are important for make to function correctly.

8. Literature Survey

How can it be possible that we have been misusing *make* for 20 years? How can it be possible that behavior previously ascribed to *make's* limitations is in fact a result of misusing it?

The author only started thinking about the ideas presented in this paper when faced with a number of ugly build problems on utterly different projects, but with common symptoms. By stepping back from the individual projects, and closely examining the thing they had in common, *make*, it became possible to see the larger pattern. Most of us are too caught up in the minutiae of just getting the rotten build to work that we don't have time to spare for the big picture. Especially when the item in question ``obviously" works, and has done so continuously for the last 20 years.

It is interesting that the problems of recursive *make* are rarely mentioned in the very books Unix programmers rely on for accurate, practical advice.

8.1. The Original Paper

The original *make* paper [1] contains no reference to recursive *make*, let alone any discussion as to the relative merits of whole project *make* over recursive *make*.

It is hardly surprising that the original paper did not discuss recursive *make*, Unix projects at the time usually *did* fit into a single directory.

It may be this which set the ``one `Makefile` in every directory" concept so firmly in the collective Unix development mind-set.

8.2. GNU Make

The GNU Make manual [2] contains several pages of material concerning recursive *make*, however its discussion of the merits or otherwise of the technique are limited to the brief statement that

> ``This technique is useful when you want to separate makefiles for various subsystems that compose a larger system."

No mention is made of the problems you may encounter.

8.3. Managing Projects with Make

The Nutshell Make book [3] specifically promotes recursive *make* over whole project *make* because:

> ``The cleanest way to build is to put a separate description file in each directory, and tie them together through a description file that invokes make recursively. While cumbersome, the technique is easier to maintain than a single, enormous file that covers multiple directories." (pp. 65)

This is despite the book's advice only two paragraphs earlier that:

> ``*make* is happiest when you keep all your files in a single directory." (pp. 64)

Yet the book fails to discuss the contradiction in these two statements, and goes on to describe one of the traditional ways of treating the symptoms of incomplete DAGs caused by recursive *make*.

The book may give us a clue as to why recursive *make* has been used in this way for so many years. Notice how the above quotes confuse the concept of a directory with the concept of a `Makefile`.

This paper suggests a simple change to the mind-set: directory trees, however deep, are places to store files; `Makefiles` are places to describe the relationships between those files, however many.

8.4. BSD Make

The tutorial for BSD Make [4] says nothing at all about recursive *make*, but it is the first the author read which actually described, however briefly, the relationship between a `Makefile` and a DAG (p. 30). There is also a wonderful quote

> ``If *make* doesn't do what you expect it to, it's a good chance the `makefile` is wrong." (p. 10)

Which is a pithy summary of the thesis of this paper.

9. Summary

This paper presents a number of related problems, and demonstrates that they are not inherent limitations of *make*, as is commonly believed, but are the result of presenting incorrect information to *make*. This is the ancient *Garbage In, Garbage Out* principle at work. Because *make* can only operate correctly with a complete DAG, the error is in segmenting the `Makefile` into incomplete pieces.

This requires a shift in thinking: directory *trees* are simply a place to hold files, `Makefiles` are a place to remember relationships between files. Do not confuse the two because it is as important to accurately represent the relationships between files in different directories as it is to represent the relationships between files in the same directory. This has the implication that there should be exactly one `Makefile` for a project, but the magnitude of the description can be managed by using a *make* include file in each directory to describe the subset of the project files in that directory.

This paper has shown how a project build and a development build can be equally brief for a whole-project *make*. Given this parity of time, the gains provided by accurate dependencies mean that this process will, in fact, be faster than the recursive *make* case, and more accurate.

9.1. Inter-dependent Projects

In organisations with a strong culture of re-use, implementing whole-project *make* can present challenges. Rising to these challenges, however, may require looking at the bigger picture.

- A module may be shared between two programs because the programs are closely related. Clearly, the two programs plus the shared module belong to the same project (the module may be self-contained, but the programs are not). The dependencies must be explicitly stated, and changes to the module must result in both programs being recompiled and relinked as appropriate. Combining them all into a single project means that whole-project *make* can accomplish this.
- A module may be shared between two projects because they must inter-operate. Possibly your project is bigger than your current directory structure implies. The dependencies must be explicitly stated, and changes to the module must result in both projects being recompiled and relinked as appropriate. Combining them all into a single project means that whole-project *make* can accomplish this.
- It is the normal case to omit the edges between your project and the operating system or other installed third party tools. So normal that they are ignored in the `Makefiles` in this paper, and they are ignored in the built-in rules of *make* programs.

  Modules shared between your projects may fall into a similar category: if they change, you will deliberately re-build to include their changes, or quietly include their changes whenever the next build may happen. In either case, you do not explicitly state the dependencies, and whole-project *make* does not apply.
- Re-use may be better served if the module were used as a template, and divergence between two projects is seen as normal. Duplicating the

module in each project allows the dependencies to be explicitly stated, but requires additional effort if maintenance is required to the common portion.

How to structure dependencies in a strong re-use environment thus becomes an exercise *in risk management*. What is the danger that omitting chunks of the DAG will harm your projects? How vital is it to rebuild if a module changes? What are the consequences of *not* rebuilding immediately? How can you tell when a rebuild is necessary if the dependencies are not explicitly stated? What are the consequences of forgetting to rebuild?

9.2. Return On Investment

Some of the techniques presented in this paper will improve the speed of your builds, even if you continue to use recursive *make*. These are not the focus of this paper, merely a useful detour.

The focus of this paper is that you will get more accurate builds of your project if you use whole-project *make* rather than recursive *make*.

- The time for *make* to work out that nothing needs to be done will not be more, and will often be less.
- The size and complexity of the total `Makefile` input will not be more, and will often be less.
- The difficulty of maintaining the `Makefile` will not be more, and will often be less.

The disadvantages of using whole-project *make* over recursive *make* are often unmeasured. How much time is spent figuring out why *make* did something unexpected? How much time is spent figuring out that *make* **did** something unexpected? How much time is spent tinkering with the build process? These activities are often though of as ``normal'' development overheads.

Building your project is a fundamental activity, if it is performing poorly so is development, debugging and testing. Building your project needs to be so simple the newest recruit can do it immediately with only a single page of instructions. Building your project needs to be so simple that it rarely needs any development effort at all. Is your build process this simple?

10. References

[1] Stuart I. Feldman, *Make - A Program for Maintaining Computer Programs*, Computing Science Technical Report 57, Bell Laboratories (Aug 1978).

[2] Richard M. Stallman and Roland McGrath, *GNU Make: A Program for Directing Recompilation*, Free Software Foundation, Inc., Cambridge, Massachusetts, USA (Jul 1993).

[3] Steve Talbott, *Managing Projects with Make, 2nd Ed.*, Nutshell, O'Reilly & Associates, Inc., Newton, MA, USA (1991). ISBN: 0-937175-18-8.

[4] Adam de Boor, *PMake - A Tutorial*, University of California, Berkeley, Beley, CA, USA (Jul 1988).

❖

Linux Expo 98

28-30 May 1998

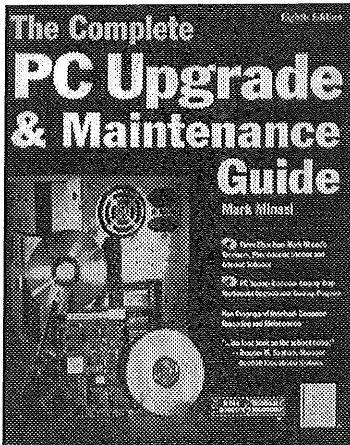
to be held at:

Duke University
Durham
North Carolina

for info see `www.li.org`

New from Addison Wesly Longman

Addison Wesley Longman



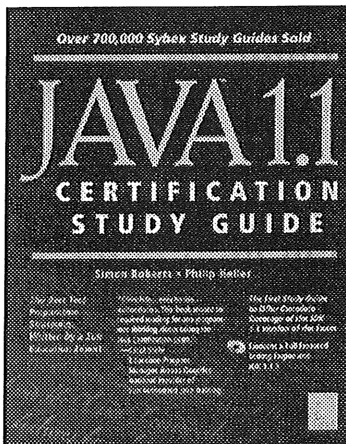
THE COMPLETE PC UPGRADE & MAINTENANCE GUIDE 8TH EDITION

MARK MINASI

New revised and updated eighth edition of this best selling title features detailed discussions of the latest PC hardware — motherboards, memory boards, sound and video cards. Also includes instructions on how to use the Internet to upgrade PCs and peripherals.

Also included are two CD-roms the first providing professional problem-solving techniques using videos from Minasi's own seminars. The second PC Tuning CD includes multimedia clips, searchable text and hyperlinks to assist in upgrading and tuning your PC.

ISBN : 0-782-2151-9 RRP $109.95



JAVA 1.1 - CERTIFICATION STUDY GUIDE

SIMON ROBERTS AND PHILIP HELLER

Every Java programmer can improve their skills with this book. Designed specifically to teach you all you need to know to pass the exam, each chapter has been technically reviewed and approved by a member of the Sun Education Team.

Inside you will find chapters that cover — Language fundamentals; Operators and assignments; Casting and conversion; Applets and HTML plus much more.

ISBN : 0-782-12069-5 RRP $79.95

AUUG Members receive a 20% Discount off the RRP

order form

To order a copy of these texts, please fill in the details below.
This form should be faxed or mailed (freepost) to Lisa Russell, Trade Marketing Co-ordinator at Addison Wesley Longman, Level 1, 2 Lincoln Street. Lane Cove NSW 2066. Ph: (02) 9428 8086 or Fax: (02) 9427 9922

Book Details

Author/Title	ISBN	Price
______	______	______
______	______	______
______	______	______
	Total less 20%	______

Order details

If you are **purchasing** the books detailed above, please give the following information. (Please note that books can also be ordered directly from bookstores.)

Book(s) price:______ Note: Prices are subject to change without notice

Postage: 1 book = $4.00, 2 or more = $7.00 Total $ ______

❒ I enclose a cheque made out to Addison Wesley Longman

❒ Please charge my credit card:

❒ Bankcard ❒ Visa ❒ Mastercard ❒ American Express ID No:______

Card Number: ______ Expiry Date ______

Signature:______

Your Name: Mr/Mrs/Ms ______

Address: ______

______

Daytime Ph: ______

Book Reviews

Advanced Oracle PL/SQL Programming with Packages

By: Steven Feuerstein
O'Reilly and Associates
1996, 661 pages + 3.5" disk
ISBN 1-56592-238-7

Reviewed by:
Matthew Dawson
<dawson.matthew.ms@bhp.com.au>
BHP Information Technology

Steven Feuerstein is an Oracle guru renowned for his articles, presentations and courses regarding the development of code that interfaces with Oracle databases. His main area of interest is PL/SQL - Oracle's procedural extension to SQL - and he wrote a 900+ page tome on the subject in 1995. Roughly a year later he has returned to his home territory, but this time focussing on a single under-utilised section of the PL/SQL language; packages.

Packages are constructs that bare many similarities to the programming concepts of both libraries and objects. They allow developers to group many related functions/procedures into a single library, but are more than just a handy repository for code. They offer developers the chance to use object-oriented design principles (particularly in regard to data encapsulation and implementation hiding) in an environment that traditionally discourages these design approaches.

In his preface Steven states his objectives as: increasing awareness of packages and their usage, distributing his software as widely as possible and making his readers more creative and effective problem solvers. A pretty tall order by most standards but he makes a valiant attempt at meeting these lofty goals.

Steven uses a conversational style of writing which makes for very easy reading. He carefully explains each concept without falling into the trap of being condescending or too simplistic for the more advanced readers. Even so this is not a book for beginners - don't even consider attacking it unless you have at least a passing familiarity with PL/SQL.

The book is broken into six sections with the first addressing the development of Oracle packages, four devoted to the suite of packages provided on the companion disk and the final section containing a series of exercises (and their solutions) to test the reader's PL/SQL knowledge. From this simple breakdown it is easy to determine the basic thrust of this publication.

Chapter one contains an introduction to the topic of packages. Steven walks the reader through a description of what a package is, the various flavours that are available and continues on to the benefits of their use. He then discusses the various components of a package; the specification (public declarations), body (implementation details and private code/data) and all of the elements that they can encapsulate. The remainder of the chapter is devoted to the mechanics of creating packages and why anyone would want to bother with them at all. I found this area of discussion both intriguing and annoying. Intriguing because I had rarely really considered packages as anything more than a place to store reusable code fragments, and annoying because I spent the remainder of the day contemplating how I could have better handled issues encountered during recent projects.

The next chapter delves into the author's 'Best Practices' for developing/designing packages. In it he expounds the philosophy that all PL/SQL development should start as packages rather than stand-alone functions/procedures i.e. that developers should approach programming with a view to create reusable application building-blocks rather than quick-and-dirty components which address only the problem at hand. The suggestions provided are practical, well explained and are focused on creating understandable, maintainable and above all useable code.

To round off the first section of the book Steven has included a chapter to illustrate the development lifecycle (or Development Spiral as he prefers to call it) of a PL/SQL function,. It is based on the creation of a function that concatenates a string to itself. Not particularly exciting as far as examples go but he somehow manages to turn this simple task into the perfect illustration of why many of the techniques from chapter two should be applied.

The remaining two thirds (or more) of the book degenerates into what is essentially a user guide for the companion disk. While the supplied packages are useful, they are merely a cutdown/unsupported version of the 'Professional' version which can be purchased from www.revealnet.com. As this full version will undoubtedly be better documented and more robust than the 'Lite' version I'm a little unsure of Steven's reasoning in devoting this large a proportion of the book to the user guide. Most serious developers planning to use his software wouldn't consider basing production programs on the unsupported version of the packages - it isn't worth the risk - so these sections are of limited use to them. Which insinuates that much of this book is intended either as an advertisement for the 'real' packages or as a play toy for hobbyists.

So did Steven reach the objectives he set for himself? I'd have to say yes. But I'm left feeling disappointed because the book didn't live up to its title. Its first

section provided me with valuable insights into this facet of PL/SQL, leaving me with high hopes for the upcoming sections. When they turned out to be little more than software documentation my attention began to wane.

The verdict? I would recommend that all Oracle developers read the first quarter of the book at least once, but I don't expect it to be a permanent feature on your bookshelf. Consider asking your workplace to buy a copy for the team...

THE PRACTICAL PERFORMANCE ANALYST PERFORMANCE-BY-DESIGN TECHNIQUES FOR DISTRIBUTED SYSTEMS

By: Neil J. Gunther
McGraw-Hill, 1998, ISBN 0-07-912946-3

Reviewed by:
Michael Paddon <Michael.Paddon@auug.org.au>

This is an ambitious book. Its fundamental thesis is that, if you want your software to meet its performance goals, you are most likely to succeed if you design in that performance from the very start. As Dr Gunther points out in his preface, however, the budget and schedule demands of the modern software project seldom leave room for essential modelling and measurement steps, especially when traditional formal methods are applied. As a consequence the performance profile of the final system is primarily accidental.

The aim of "The Practical Performance Analyst" is to give the reader sufficient tools for streamlined modelling and measurement. This is a straightforward tradeoff between the resolution of the analysis and the resulting cost in time and resources. In most cases, a software designer does not need an exact answer, just an understanding of the critical performance aspects of an architecture and enough guidance to choose intelligently between different design options.

Section one of the book, "Foundations", starts by discussing the nature of time, both in the real world and in the measurement of events within a computer. This naturally leads on to discussion of fundamental time based metrics. This is followed by "Queuing Theory For Those Who Can't Wait", which is the lynchpin of the entire work. Dr Gunther happily throws all the hard stuff (stochastic modelling, probability theory and simulation) out the window, and replaces it all with a system of basic queue types and simplified formulae founded on averages. This works surprisingly well, and the math turns out to be trivial. Best of all, a simple one page summary at the end of the chapter makes it all too easy. A more detailed discussion of modelling the real world with systems of queues then provides the final piece of the puzzle. The section ends with a chapter covering commonly available distributed performance management protocols and tools.

Section two, "Applications", focuses on the practical. It kicks off with a brief overview of currently available commercial parallel architectures, and the types of applications (especially databases) that are driving the development of such machines. An analysis of the various parallel architectures follows, with emphasis on measuring their effectiveness, both in terms of pure parallel speedup and real world workloads. SMP architectures, as the dominant contemporary multiprocessor, are modelled and analysed in fine detail. Whilst this is of some help to a software engineer like myself, the next target of discussion, a similarly detailed analysis of client-server architectures, is invaluable. Client-server methods are so ubiquitous in modern software and the techniques presented gave me a new understanding of performance issues and limitations that I need to deal with daily. The section ends with an analysis of Web servers which, not surprisingly, shows fundamental performance bottlenecks in HTTP, but also examines the effects of different design decisions in the servers themselves.

Finally, section three, "Innovations", presents some more complex ideas and analysis techniques. In particular, the math gets a bit more complex and a bit of calculus starts popping up. Dr Gunther, however, does an admirable job at explaining what is going on in plain English, so you can quite happily ignore the integrals, and still come away with a solid understanding of the concepts. The section begins with an examination of stable and unstable states, non linear system response and transient behaviour. This quickly becomes a very formal exploration of path integrals and their applicability in large population systems. In simple terms, this is all about high powered tools to analyse large scale systems. All this math is then thrown at typical packet switched and circuit switched network problems in a frenzy of no-holds-barred serious performance analysis. The last chapter of the section takes a complete break from the preceding material, and models the non-linear performance degradation effects of an SMP architecture. I got the feeling that this got put at the end, simply because it didn't slot in anywhere else neatly.

The book also includes a source, documentation and sample code for a piece of performance analysis software called PDQ, or Pretty Damn Quick. PDQ is a queuing circuit solver that uses the averaging techniques presented by the book to rapidly calculate performance metrics without resorting to simulation. This tool is used throughout the book to calculate real numerical answers to problems, so it looks like it will be pretty useful in the real world as well.

As you can see by now, this is not a lightweight book. Effective performance measurement and analysis are demanding tasks requiring a deep understanding of what is actually happening behind the models and metrics. "The Practical Performance Analyst" does not claim to make this easy. However, it places in the reader's hands the tools and techniques needed to do the job properly, fast and as accurately as required. Furthermore, the book is superbly and clearly written. Anyone with a typical computing background can pick up this tome today and start applying its lessons tomorrow. For such a complex subject, that is quite an achievement.

I don't often find myself unreservedly recommending a book. The fact is that "The Practical Performance Analyst" should be on every programmer's and sysadmin's bookshelf. Have a look at it, and I think you'll agree.

THE JAVA LANGUAGE REFERENCE, 2ND EDITION

By Mark Grand
O'Reilly & Associates, 2nd Edition July 1997
492 pages, $32.95 US, ISBN: 1-56592-326-X

Reviewed by:
David Hook <dgh@aba.net.au>

Mark Grand's resume states that he spends a great deal of time teaching the Java programming language, and primarily this book is an introductory text, which probably would be useful, up to a point, in enabling someone to learn the in, outs, and subtleties of the Java programming language and its primary API, the java.lang package. Each chapter is divided into sections and each section includes a list of cross references to other parts of the book making the text a very easy one to get around in.

The book is roughly divided into two parts, the first part covers the types and syntax of the language, a look at object oriented programming - Java style, the use of Threads, classes, and interfaces, and a reasonably well documented look at issues such as synchronisation, the ins and out of flow control, and exception handling. All the fundamentals appear to be covered, and the first part of the book also includes a brief look at how applications and applets are put together.

The second half of the book is largely a repeat of what you would find in the JavaDoc documentation for java.lang package. While it is cross referenced with the rest of the book, I could not avoid thinking that the space would have been better taken up with a basic look at the input/output features of the io package and the standard utilities. At the moment the only thing covered in the book is println!

In what it covers the book is very thorough, and while it is certainly not possible to cover all the Java programming language in one book, I do believe the book is of limited use to someone wishing to program anything "real" in Java. Indeed, on the back cover the publishers mention that the book should be used in conjunction with two other books, and it is my feeling that this book would be one of those which would rapidly migrate to the back of the bookshelf. If you are already using Java and have one or two books, this probably is not a good book to buy. If you are interested in learning Java, this book is worth considering, although if you are looking for a broader introduction to the language and its APIs, I would recommend having a look at "The Java Programming Language" by Ken Arnold and James Gosling first.

Meet the AUUG-Exec

MICHAEL PADDON

The first computer I ever got to lay my hands on was a PDP 11/04 at the tender age of thirteen. This machine came complete with a mark sense card reader, a line printer and a shiny new copy of the MONECS operating system. This was a great system to learn with, despite the fact that the entire operating system fit on a single eight inch floppy and was only marginally more functional than MSDOS. The Fortran compiler was the only one worth using but, despite that, I retained my sanity. All this, and the machine only tried to kill me twice!

Then I discovered machine code and my world changed forever. I actually worked out how computers tick. Shortly thereafter I purchased my very own TRS-80 and settled into Z-80 assembler heaven. Actually, writing, assembling, linking and crash testing machine code on a computer with just

about no debugging tools and a tape drive as mass storage isn't fun...but I didn't know that back then.

University led to two new important things in my life... girls and real operating systems. Meeting Unix and C for the first time is a satori when all you've seen are horrible little toy computer program loaders. Quite by good fortune, I'd ended up at Melbourne Uni, and had the benefit of finding myself in a hotbed of local Unix hacking. I also got to play with real computers, like vaxen. Wow! And later on, these weird things called Sun workstations. By the way, did I mention the girls? Who says computer science is boring?

Armed with a fresh new BSc(Hons), I signed up as a staff member at the Uni and spent two years researching and building user interface technology. I also spent way too much time posting netnews and organising aus.sf dinners. Around that time, the Uni got hooked up to this thing that one day would be called the Internet. A whole new world.

Naturally it was only a matter of time before I sold out to the commercial world for the big bucks. Yeah. Two years at DEC, nearly three at an underfunded startup called Iconix, and then two years at Kodak preceded my current lifestyle as technical director of Australian Business Access, an internet commerce startup. See, I don't learn.

I've been involved with AUUG since my university days, and actually got suckered into presenting my first paper at the 1989 winter conference. I can't begin to describe what influential and inspirational environment AUUG provided for me back then. Suffice it to say that I and my peers were all improved and enriched by being involved.

Around 1990 and 1991, I started to get really pissed off by what I saw as the dilution of AUUG's strengths by short term and short sighted interests. There even seemed to be a danger of a schism into two groups, which would have been an unmitigated disaster. After one too many of my complaints, some wag suggested I should actually do something, and by 1992 I was elected to the executive committee. Three years of that, followed by three as president have given me ample opportunity to "fix" things. You get to be the judge of whether I've done any real good.

Nowadays? Professional interests are operating systems, security, protocols, the usual grab bag. One day I'm going to finish that perfect OS. In my spare time (hah!) I study Japanese (which gives me an excuse for the manga and anime addiction), read novels (mostly SF), watch movies (with a weakness for Hong Kong cinema), race the RX5, fix the RX5, torment my cat and sleep. Somewhere along the line I wound up married to a wonderful lady, Linda, who is way too nice for a guy like me. Go figure.

❖

Luigi Cantoni

I have been involved with Unix since about 1981 first with Microsoft (OH NO I said that company name) XENIX sys III. This is the version SCO first started with when they said "We think this is a good product" and Bill said "Go forth you sinners". This was an interesting version as it had a split Kernel. One half was all the screen and disk I/O processing and was done on a 80186 (remember one of those). The other half was the true kernel and it ran on a 80286 processor. Multi processor back in 81? Since then I have predominately used various versions of SCO although I have had brief stints with other variations like DGUX, Motorola, Sun AIX etc.

I have predominately worked for small firms where basically I have been the entire computer dept. This has meant that I have had to do all sorts of System work and administration even though my passion is accounting and business applications. I remember early at a WAUUG meeting when we where all discussing what we did. As I was near the end I was able to state I was a USER of Unix. I was not involved with a Software house or Vendor and was not from an Academic institution. That is not true now as I am in charge of development with a team of programmers. I both program predominately in 'C' and a language tailored to our business applications and do a great deal of shell scripting.

I have been involved with several aspects of the building industry. Several engineering type firms and now am heavily involved in the quick service (that's fast food to humans) industry. Some of my clients include Hungry Jacks, KFC, Dominoes, Burger King and several smaller operations. Several overseas firms are also looked after and that means 24hr a day 7 days a week being on call to these clients.

I am proud to say I have several sites that run my applications that are still on their original hardware with the same software for over 10 years and require less than a few hours a year of systems/application support. Who said Unix requires constant systems support. All my working life has been in computing and almost all of that with Unix. I fully expect that to continue.

PS I'm writing this while waiting for my name to be called for Jury Duty.... No one loves me I can go home instead.

❖

AUUG BOOK CLUB & PRENTICE HALL AUSTRALIA

20% DISCOUNT TO AUUG MEMBERS

Please send me the following book/s on 30-Day approval

	Title	Format	RRP	Price
☐	Unix Unleashed Internet Edition - *Burk et al Sams Publishing* ISBN 0672312050	Cloth	~~$109.95~~	$ 87.95
☐	HP-UX Systems Administration Handbook and Toolkit - *Poniatowski Prentice Hall* ISBN 0139055711	Paper	~~$89.95~~	$71.95
☐	Mobile IP The Internet Unplugged - *Solomon Prentice Hall* ISBN 0138562466	Cloth	~~$49.95~~	$39.95
☐	Timebomb 2000 - *Yourdon Prentice Hall* ISBN 0130952842	Paper	~~$19.95~~	$15.95
☐	Cisco IOS Configuration Fundamentals - *Cisco Systems Cisco Press* ISBN 1578700442	Cloth	~~$119.95~~	$95.95
☐	Internetworking Technologies Handbook - *Cisco Systems Cisco Press* ISBN 1562056034	Cloth	~~$79.95~~	$63.95

AUUG members receive 20% discount below recommended retail price

Mail Fax Phone or Email your order to:

ATTN: Jan Blenkinsop
Prentice Hall Australia, Marketing Department
Locked Bag 507, Frenchs Forest NSW 2086
Tel:(02) 94542211 Fax:(02) 9453 0117 Email: jan_blenkinsop@prenhall.com.au

Name: ______________________ Position: ______________

Company: ______________________

Address: ______________________

______________________ Telephone: ______________

PLEASE SEND MY BOOK/S ON 30-DAY APPROVAL

☐ Enclosed cheque for $ _______ (Payable to'Prentice Hall Australia')

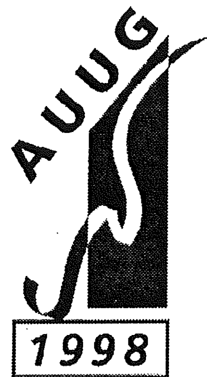
☐ Charge to me **OR** ☐ Company purchase Order No. __________

Please charge my: ☐ Bankcard ☐ Visa ☐ MasterCard ☐ AMEX

Expiry Date: ______________ Credit Card No: ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

Signature: ______________________

Call For Papers



AUUG98 Conference
September 3-5, 1997
Sydney Hilton Hotel,
Sydney, New South Wales,
Australia

THEME: "OPEN SYSTEMS: THE COMMON THREAD"

The 1998 AUUG winter conference will be held at the Sydney Hilton Hotel, New South Wales, Australia, between September 16th and 18th.

The conference will be preceded by two days of tutorials, on September 14th and 15th.

The program committee invites proposals for papers and tutorials relating to:

Technical aspects of Unix and Open Systems

New developments in open software systems, languages and applications

Networking, Internet (including the World Wide Web) and Security

Business and Management Experience and Case Studies

The theme of this years conference is "Open Systems: The Common Thread". The program committee will interpret the theme very broadly with the aim of highlighting the breadth of applicability for Open Systems. As always, papers and tutorials with a strong technical flavour are particularly welcome.

Presentations may be given as tutorials, technical papers, or management studies. Technical papers are designed for those who need in-depth knowledge, whereas management studies present case studies of real-life experiences in the conference's fields of interest. Tutorials may be either 1/2 day or full day and have a strong practical focus.

All presentations must be accompanied by a written paper for the conference proceedings.

Speakers may select one of two presentation formats:

Technical presentation:
a 25 minute talk, with 5 minutes for questions;

Management presentation:
a 20-25 minute talk, with 5-10 minutes for questions (i.e. a total 30 minutes);

Panel sessions will also be timetabled in the conference and speakers should indicate their willingness to participate, and may like to suggest panel topics.

Tutorials, which may be of either a technical or management orientation, provide a more thorough presentation, of either a half-day or full-day duration.

Representing the largest Unix and Open Systems event held in Australia this conference offers an unparallelled opportunity to present your ideas and experiences to an audience with a major influence on the direction of computing in Australia.

SUBMISSION GUIDELINES

Those proposing to submit papers should submit an extended abstract (1-3 pages) and a brief biography, and clearly indicate their preferred presentation format.

Those submitting tutorial proposals should submit an outline of the tutorial and a brief biography, and clearly indicate whether the tutorial is of half-day or full-day duration.

SPEAKER INCENTIVES

Presenters of papers are afforded complimentary conference registration.

Tutorial presenters may select 25% of the profit of their session OR complimentary conference registration. Past experience suggests that a successful tutorial session of either duration can generate a reasonable return to the presenter.

IMPORTANT DATES

Abstracts/Proposal Due:	May 15, 1998
Authors notified:	June 5, 1998
Final copy due:	August 7, 1998
Tutorials:	September 14-15, 1998
Conference:	September 16-18, 1998

Proposals should be sent to:

AUUG Inc.
PO Box 366
Kensington NSW 2033
AUSTRALIA

Email: auug98@auug.org.au

SAGE-AU Sixth Annual Conference and General Meeting

Tuesday 7/7/1998 to Friday 10/7/1998
Old Parliament House
Canberra, ACT,
Australia

Call for Papers and Tutorials

The System Administrators Guild of Australia (SAGE-AU) will be hosting its sixth annual conference in conjunction with its 1998 annual general meeting.

The annual SAGE-AU Conference, Tutorials and AGM provides a forum for Systems Administrators, Systems Managers, Network Administrators, Developers of Systems Administration Software and Managers of such groups to meet and share their knowledge and experiences.

SAGE-AU'98 is hereby calling for papers and tutorial presentations on any and all topics related to system administration.

Deadlines

Applications to present tutorials and papers must reach the organisers by April 3, 1998.

To be included in the conference proceedings, papers must reach the organisers by June 19, 1998.

Conference Details

SAGE-AU'98 will be a 4 day conference running from Tuesday July 7, 1998 to Friday July 10, 1998.

The first two days (Tuesday & Wednesday) will be dedicated to tutorials on tools and techniques to aid system administration.

The AGM will be held at the end of the third day (Thursday). All other times will be allocated to presentations or discussions.

A conference dinner will be held on the Thursday evening.

The conference will feature a small trade show on the third and fourth days, focusing on system administration tools and information.

Papers

Timeslots are available for 15, 30, 45 and 60 minute presentations. 5-10 minutes should be reserved for questions from the audience.

15 minute timeslots are less formal and are to allow people to talk briefly about some topic of interest or problem without having to prepare a formal paper (Work in Progress).

People presenting a 30+ minute talk will receive free conference registration.

People presenting a 15 minute talk will receive a 50% discount on the conference registration fees.

If you wish to present a paper, send an abstract to the address below by the due date. Please indicate whether you are asking for a 15, 30, 45 or 60 minute timeslot.

Abstracts should be 100 -- 200 words in length. Papers should have a technical orientation and should not contain advertising.

People giving 30+ minute presentations will be expected to provide a paper for inclusion in the conference proceedings.

Tutorials

Tutorial sessions will be either half day or full day in duration. People wishing to present tutorials should submit an abstract of the material they wish to present and an indication of whether they require a half day or a full day timeslot. Tutorials should be run in lecture format. Suggested topics include:

- Computer and Network Security/Network Authentication
- PC/Apple/Unix/Mainframe Interoperability
- NFS/Automount/AMD Configuration and Operation
- Perl/Java/Tcl/Python
- Sendmail/Qmail/smapd/Anti-SPAM
- WWW Cache/Router Config/Firewall Setup/Squid
- NT/Win95 Administration

Tutorial presenters will be paid $500 for a half day tutorial and $1000 for a full day tutorial and will receive free conference registration. SAGE-AU will reimburse tutorial presenters for reasonable costs of handout materials or will print them on your behalf.

As with papers, tutorials should have a technical orientation and should not contain advertising.

EXHIBITION/TRADE SHOW

On the third and fourth days of the conference (Thursday and Friday) SAGE-AU'98 will host a small, technically orientated trade show focusing on system administration tools and information.

If you or your company are interested in participating in the trade show please contact the organisers for details.

REGISTRATION

Conference registration includes one ticket to the Conference Dinner and Conference and Tutorial registration includes Lunch and Refreshments. Additional tickets to the Conference Dinner may be purchased.

Non-members who register for SAGE-AU'98 at the non-member rate and successfully apply for membership of SAGE-AU will have their first year's membership fee waived.

Conference registration forms will be available in mid May 1998.

Registration forms for tutorials will be available approximately six weeks before the conference date.

Early Registration is considered when registration form and payment has reached SAGE-AU by COB on 19th of June 1998.

TRAVEL

To encourage interstate attendees, SAGE-AU offers members a travel discount off registration for interstate travellers (Qld/Vic/Tas/WA/SA/NT).

ADDRESSES

Send all enquiries regarding the conference as well as tutorial and paper abstracts to:

SAGE-AU'98
GPO Box 2984
Sydney NSW 2001
Australia

E-Mail: conference@sage-au.org.au

Requests for general information about SAGE-AU and membership applications should be addressed to:

WWW: `http://www.sage-au.org.au/`

Email: secretary@sage-au.org.au

Fax: 0500 544 488 (Attn: David Conran)

Or alternatively,

Secretary
SAGE-AU
GPO Box 2984
Sydney NSW 2001
Australia

WWW PAGE

A web page for the conference is

`http://www.sage-au.org.au/conf.html`

TELLURIAN

Tellurian Pty Ltd

Come to us if you need seriously capable people to help with your computer systems. We're very good at what we do.

- Unix, Macintosh and Windows experts
- Legacy system re-engineering and integration
- System management and support
- Internet access

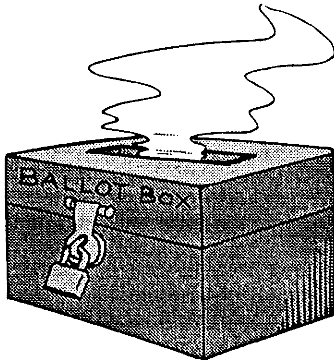
Our two current major projects:

- Support and development of an integrated environment covering applications running on IBM3090, DEC Alpha, SCO Unix and Nortel switches. Just imagine the cost benefits of supporting over 500 concurrent users on four little 486 and Pentium PC's.
- From the ground-up implementation of MFC and Windows API on Apple Macintosh. We've got our client's Windows MFC application running, bug-for-bug, on Apple Macintosh.

Tellurian Pty Ltd
272 Prospect Road
Prospect SA 5082

(08) 8408 9600
www.tellurian.com.au
sales@tellurian.com.au

We want you...



to nominate for a position on the AUUG Management Committee.

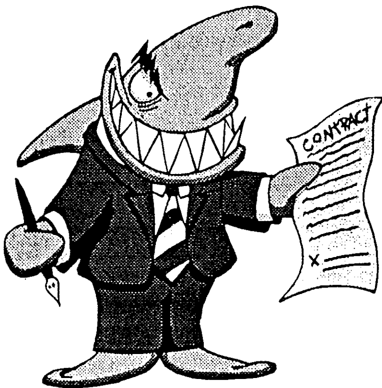
Help make AUUG the kind of organisation you want it to be – nominate for a position on the AUUG Management Committee! The call for nominations and a sample nomination form can be found on the next few pages. The nomination form should be returned to AUUUG by the 14th of April.

You need to be nominated by three voting members of AUUG (that is, either individual members or institutional members), and you must be an individual member yourself.

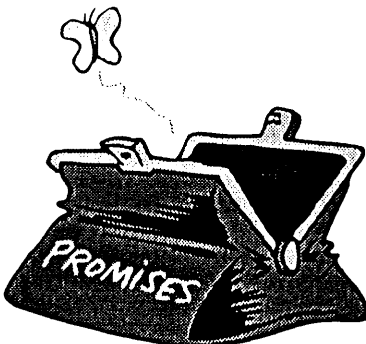
If you want to know more about serving on the Management Committee, e-mail the current committee at:

auugexec@auug.org.au

What? You can't find three members to nominate you?



Send in your nomination form anyway – we'll find someone to sign it.



And don't forget your 200 word policy statement!

AUUG Incorporated
1998 Annual Elections
Nomination Form

We,

(1) Name: ______________________________ AUUG Member #:________ and

(2) Name: ______________________________ AUUG Member #:________ and

(3) Name: ______________________________ AUUG Member #:________

being current financial members of AUUG Incorporated do hereby nominate:

______________________________ for the following position(s):

(Strike out positions for which nomination is not desired. Each person may be elected to at most one position, and election shall be determined in the order shown on this nomination form.)

President

Vice President

Secretary

Treasurer

Ordinary Management Committee Member (5 positions)

Returning Officer

Assistant Returning Officer

Signed (1) ______________________________ Date ____________

Signed (2) ______________________________ Date ____________

Signed (3) ______________________________ Date ____________

I, Name:______________________________ AUUG Member #: ________

do hereby consent to my nomination to the above position(s), and declare that I am currently a financial ordinary member of AUUG Incorporated.

Signed ______________________________ Date ____________

AUUG Incorporated 1998 Annual Elections Call for Nominations

Nominations are invited for the following positions within AUUG Incorporated:

President
Vice President
Secretary
Treasurer
Ordinary Management Committee Member (5 positions)
Returning Officer
Assistant Returning Officer

Nominations must be made in writing and must be signed by the nominee and three (3) financial voting members of AUUG Incorporated, and must state which position(s) are sought by the nominee. The nominee must be a financial ordinary member of AUUG Incorporated, and can nominate for any or all of the above positions. While any ordinary member may be nominated to more than one position, no person may be elected to more than one position. Election to positions is determined in the order shown above.

A sample nomination form can be found on the previous page.

Nominees may include with their nomination a policy statement of up to 200 words. This word count will not include sections of the statement stating, in point form, the name of the nominee and positions held on, or by appointment of, the AUUG Management Committee or positions in AUUG Chapters.

Policy statements that exceed the word limit shall be truncated at the word limit when included in the ballot information.

Nominations must be received by the Secretary of AUUG Incorporated by the 14th of April 1997, and may be lodged by one of the following methods:

(1) by post to:
The Secretary
AUUG Incorporated
PO Box 366
Kensington, NSW, 2033

(the nomination must be received no later than April 16th and must be postmarked no later than 12 noon on April 14th 1997).

(2) by hand to:
The Secretary (David Purdue) OR
The AUUG Incorporated Secretariat
no later than 5pm on April 14th 1997.

(3) by FAX to:
The Secretary (fax to (02) 9904 7057, marked Attn: David Purdue) OR
The AUUG Incorporated Secretariat (fax to (02) 9332 4066)

no later than 5pm on April 14th 1997.

David Purdue
Secretary
AUUG Incorporated

AUUG Incorporated Election Procedures

These rules were approved by the AUUG Inc. Management Committee on 14/12/1994.

1. NOTICE OF ELECTION

The Returning Officer shall cause notice of election to be sent by post to all financial members no later than March 15 each year.

2. FORM OF NOTICE

The notice of election shall include:

(a) a list of all positions to be elected, namely:
- President
- Vice President
- Secretary
- Treasurer
- Ordinary Committee Members (5)
- Returning Officer
- Assistant Returning Officer

(b) a nomination form;

(c) the date by which nominations must be received (in accordance with clause 21(2) of the Constitution, this date is 14 April);

(d) the means by which the nomination form may be lodged;

(e) a description of the format for a policy statement.

3. POLICY STATEMENT

A person nominated for election may include with the nomination a policy statement of up to 200 words. This word limit shall not include sections of

the statement stating in point form the nominee's name, personal details and positions held on, or by appointment of, the AUUG Management Committee and chapters.

Policy statements exceeding the word limit shall be truncated at the word limit when included in the ballot information.

The Returning Officer may edit policy statements to improve readability, such edits being limited to spelling, punctuation and capitalisation corrections and spacing modifications.

Use of the UNIX wc program shall be accepted as an accurate way to count words.

4. RECEIPT OF NOMINATIONS

In accordance with clause 21(2) of the Constitution, nominations shall be received by the Secretary up until April 14. A nomination shall be deemed to have been received by the due date if one of the following is satisfied:

- it is delivered by post to AUUG Inc's Post Box, the AUUG Secretariat's Post Box or the AUUG Secretariat's street address no later than 2 business days after April 14 and is postmarked no later than 12 midday on April 14;
- it is delivered by hand to the Secretary or the AUUG Inc Secretariat no later than 5 pm on April 14;
- it is transmitted by facsimile to the Secretary or the AUUG Inc Secretariat no later than 5 pm on April 14.

5. REQUIREMENT FOR A BALLOT AND DUE DATE

In accordance with clause 21(5), no later than May 1, the Secretary

- shall advise the Returning Officer of all valid nominations received;
- and if a ballot is required, shall advise the Returning Officer of a date no later than May 15 for the ballot for all contested election.

In accordance with clause 42(3), the due date for return of ballots shall be 4 weeks after the date advised above.

6. FORM OF BALLOT PAPER

The ballot paper shall contain:

- details of all positions for which the number of nominations exactly equals the number of positions to be filled;
- for each position for which a ballot is required, the names of all persons seeking election to that position, except those already elected to a higher position, with a square immediately to the left, for the elector to place a voting preference;
- instructions on how to complete the ballot paper;
- instructions on how to return the ballot paper;
- a brief description of how the ballot is to be counted.

The ballot paper shall not contain any identification of existing office-bearers.

The ballot paper shall be accompanied by a copy of all policy statements submitted by all persons nominated, including any persons elected unopposed. These policy statements may be truncated or modified as outlined in 3.

7. METHOD OF VOTING

Voting for each position shall be by optional preferential vote. The number "1" must be placed against the candidate of the elector's first preference, and a number other than "1" against any or all of the other candidates. Preferences shall be determined by the numbers placed against other candidates, which must be strictly monotone ascending to count as preferences.

A vote shall be informal if:

- it does not have the number "1" against exactly one candidate.

8. SECRECY OF BALLOT

The ballot paper shall be accompanied by two envelopes, which may be used by the elector to ensure secrecy. On completion of the ballot paper, the paper may be placed inside the smaller envelope. This envelope is then placed inside a second envelope. The elector must then sign and date the outer envelope, making the following declaration:

"I, ________________________________,
member number __________, declare that I am entitled to vote in this election on behalf of the voting member whose membership number is shown above, and no previous ballot has been cast on behalf of this voting member in this election."

9. RETURNING BALLOT

To be considered to have been returned by the due date, the ballot paper together with declaration as above must be returned by one of the following means:

- it is delivered by post to AUUG Inc's Post Box, the AUUG Secretariat's Post Box or the AUUG Secretariat's street address no later than 2 business days after the due date and is

postmarked no later than 12 midday on the due date;

- it is delivered by hand to the Returning Officer or the AUUG Inc Secretariat no later than 5 pm on the due date.

10. METHOD OF COUNTING

Where there is an election for a single position, the votes shall be counted by the preferential method. Where there is more than one position to be filled, the votes shall be counted by the modified preferential Hare Clark system described in Schedule 1.

11. METHOD OF ELECTION

A person may be elected to only one position. Elections shall be counted in the order of positions described in 2(a). When counting ballots, any person previously elected shall be deemed withdrawn from that election, and all ballot papers shall be implicitly renumbered as though that person was not included.

12. NOTIFICATION OF RESULT

In accordance with clause 42(7) of the Constitution, the Returning Officer shall advise the Secretary in writing of the result no later than fourteen days after the due date. The Returning Officer shall advise all candidates for election of the result no later than fourteen days after the due date. The Returning Officer shall advise the AUUGN Editor in writing of the result no later than fourteen days after the due date. The AUUGN Editor shall include the results in the first issue of AUUGN published after receiving the results from the Returning Officer.

13. PUBLICATION OF THESE RULES

The Returning Officer shall advise the AUUGN Editor of the current rules, and the AUUGN Editor shall cause the current rules to be published in the first issue of AUUGN published on or after 1 January each year. Where no issue of AUUGN has been posted by February 28 in any calendar year, the Returning Officer shall cause the current rules to be distributed with the notice of election.

14. OCCASIONAL VARIATION FROM THESE RULES

Subject to the Constitution, the Management Committee may authorise occasional variations from these rules. Such variations shall be advised in writing to all members at the next stage in the election process in which information is distributed to members.

15. EXECUTION

Where these rules require the Returning Officer to carry out an action, it shall be valid for the Returning Officer to delegate execution to the Secretariat from time to time employed by the Management Committee.

16. RETENTION OF BALLOT PAPERS

The Secretary shall retain that ballot papers and member declarations (as specified in 8) until the AUUG AGM of the calendar year following the year of the election, unless a general meeting of AUUG directs the Secretary to hold them for a longer period.

Schedule 1

1. Each ballot paper shall initially have a value of one.
2. The value of each ballot paper shall be allotted to the candidate against whose name appears the lowest number on the paper among those candidates not elected or eliminated. If there is no such candidate (i.e. the ballot paper is exhausted) the ballot paper shall be set aside.
3. A quota shall be calculated by dividing the number of formal votes by one more than the number of positions remaining to be elected, and rounding up to the next whole number.
4. If any candidate is allotted a total value greater than the quota, that candidate shall be declared elected, and the ballot papers allotted to that candidate shall be assigned a new value by multiplying their previous value by the excess of the candidate's vote above the quota divided by the candidate's total vote. This new value shall be truncated (rounded down) to 5 decimal places. Ballot papers that subsequently have a value of zero shall be set aside. Steps 2 and 3 shall then be repeated.
5. If no candidate is allotted a total value greater than the quota, the candidate who is allotted the lowest total value among those candidates not elected or eliminated shall be eliminated. Steps 2 and 3 shall then be repeated.
6. Where

   (a) two or more candidates declared elected at the same stage of counting according to Step 4 have an equality of votes, and it is necessary to determine which is deemed elected first,

   or

   (b) a candidate is required to be eliminated under Step 5, and two or more candidates have an equally low vote,

   the Returning Officer shall return to the immediately preceding stage of counting and

   (i) in the case of candidates elected, deem first elected the candidate with the highest vote at the immediately preceding stage, and

   (ii) in the case where a candidate is to be eliminated, eliminate the candidate with the lowest vote at the immediately preceding stage.

   Where an equality of votes still exists at the immediately preceding stage, the Returning Officer

shall continue proceeding to preceding stages until a result can be determined.

In the event that candidates have maintained an equality of votes throughout the entire counting process, the Returning Officer shall determine which candidate is to be determined first elected or to be eliminated by lot in the presence of the Assistant Returning Officer.

❖

Returning Officer's Report AUUG Rules Ballot, September 1997

Chris Maltby
AUUG Returning Officer

Ballot Envelopes received	139
Excluded votes	9
Student Members	6
Duplicate Inst.	1
Unknown voter	1
Blank declaration	1
Informal (no ballot paper encl.)	1
Total not counted	10
In Favour	129
Against	0
Total	139

I declare the proposal to change the AUUG Rules to have been carried unanimously.

❖

Chapter News:

Canberra

The next chapter meeting will be held on the 10th of March. Ken Day from the Computer Crime Squad of the Australian Federal Police will be talking about Computer Security and the law. The meeting will be held at 7.30pm in Room LG102, the John Dedman ***Building, Australian National University.***

❖

Chapter News:

AUUG-NSW

AUUG NSW Presidents Report 1997

1997 was a mixed bag as far as AUUG NSW was concerned. The year started with a delayed summer technical conference after the original conference which was planned for Bathurst was cancelled due to poor registration numbers. This lead to a rethink on the whole conference issue resulting in a move to a less grand and much cheaper approach. It was also decided to experiment with a Saturday date, to enable people with heavy work commitments to attend. The resulting conference, while small, was well received and the approach was continued with the 1998 summer conference.

Meetings were held on the 3rd Thursday of each month, starting at 7:00PM. After quite a bit of searching, the Wesley conference centre was finally chosen as the venue, and feedback on the facility from members has been positive. Attendance at meeting has been growing slowly, with the most popular talks being on SPAM (Pauline Van Winson), LINUX (Jon "Mad Dog" Hall), Java (Larry Weber), SSH (Charlie Brady) and router technology (Andrew McCrae). January's meeting was a barbecue in Lane Cove National Park, which was great fun, although (again) not particularly well attended.

Attendance at meetings ranged between 10 and 30 people over the 12 months, not entirely satisfactory out of more than 300 AUUG members in the Sydney area. The incoming committee should focus efforts on increasing meeting attendance. The Christmas party was a particular disappointment with only around 10 people attending. The closeness of the date to Christmas may have adversely affected attendance.

The 1998 summer conference is currently being organised and will be held on Feb 20 and 21 at the Wesley Centre in conjunction with the 1998 AGM.

Finding good speakers who have a high standard of technical excellence is hard, and has been getting harder. Finding people who can help with mundane tasks on the Committee is also hard. 1998 will be a challenge to the new Committee, which I wish them every success in overcoming.

❖

AUUG Local Chapter Meetings 1998

CITY	DATES	LOCATION	OTHER
BRISBANE	24 February 31 March 26 May 30 June 28 July 25 August 29 September 27 October 24 November	Inn on the Park 507 Coronation Drive Toowong	For further information, contact the QAUUG Executive Committee via email (qauug-exec@auug.org.au). The techno-logically deprived can contact Rick Stevenson on (07) 5578-8933. To subscribe to the QAUUG announcements mailing list, please send an e-mail message to: <majordomo@auug.org.au> containing the message "subscribe qauug <e-mail address>" in the e-mail body.
CANBERRA	10 March 14 April 12 May 9 June 14 July 11 August 8 September 13 October 10 November 8 December	Australian National University	
HOBART	Each month, although dates can vary. Often will fit in with the schedule of a speaker should one be available.	University of Tasmania	
MELBOURNE	18 February 18 March 15 April 20 May 17 June 15 July 19 August 21 October 18 November 16 December	Various. For updated information See: http://www.vic.auug.org.au/auugvic/av_meetings.html	The meetings alternate between Technical presentations in the odd numbered months and purely social occasions in the even numbered months. Some attempt is made to fit other AUUG activities into the schedule with minimum disruption.
PERTH	18 February 18 March 15 April 20 May 17 June 15 July 19 August 21 October 18 November 16 December	The Victoria League 276 Onslow Road Shenton Park	Meeting commences at 6.15pm
SYDNEY	19 March 16 April 21 May 18 June 16 July 20 August 15 October 19 November 17 December	The Wesley Centre Pitt Street Sydney 2000	The February meeting will be replaced by the summer conference on 21 February.

*** All dates are subject to change.**
Up-to-date information is available by calling AUUG on 1-800-625-655.

UNIX Traps & Tricks

Sub-Editor: Matthew Dawson
<dawson.matthew.ms@bhp.com.au>

Hi Everyone! Welcome to another issue of UNIX Tricks & Traps - a column designed to provide insights into how your fellow AUUG members make their day-to-day usage of UNIX easier.

You may have noticed the new name at the top of this page. Since Günther (the previous UT&T editor) has taken over as editor of AUUGN he has been unable to spend as much time on this column as he would of liked. As a result he has decided to hand over the reins to myself, so that he has more time to devote to his editorial role.

I'd like to thank this month's contributors, Graham Jenkins and the ever present (in AUUGN at least) David Purdue, for their useful tips. This is also the perfect chance to plead for more UT & T contributions - the cupboard is looking very bare at the moment. All of you have the qualifications necessary to supply a useful tip - you use UNIX - so please take the time to share some of the secrets that make your use of this O/S more enjoyable. If you have found any items in UT & T useful in the past, it is the least you can do to return the favour.

Anyway, I suppose its time to hop off the soap box and get back to our regularly scheduled program...

❖

TIDYPATH

From: David Purdue <David.Purdue@Aus.Sun.COM>

For various reasons that I do not want to go in to here, I do not have full control of my .login and .cshrc files. The result of this is that various scripts that are out of my control add directories to my path. Thus I end up with a path that contains many redundant directories and is so long that the "which" command refuses to search it.

And so I wrote tidypath. Remember that it is only the first occurrence of a directory in your path that makes any difference - if the shell did not find a program the first time it searches that directory, it almost certainly won't find it the second. So tidypath takes the value of the PATH environment variable and strips from it any directory that is repeated. You end up with a shorter path that has the same effect.

Use it like this:

In csh:	% setenv PATH \`tidypath $PATH\`
In Bourne-like shells:	$ export PATH=\`tidypath $PATH\`

I tried to write tidypath as a shell script, but after ten minutes could not get the right combination of commands and so gave in and wrote it in C. I imagine there is a 5 line equivalent in perl, but I don't have a week to find, install and learn perl.

```
/*********************************************************************/
/*                                                                   */
/*    tidypath.c - tidy up a long $PATH.                             */
/*                                                                   */
/*    Usage:  csh:            % setenv PATH `tidypath $PATH`         */
/*            sh, ksh:        $ export PATH=`tidypath $PATH`         */
/*                                                                   */
/*    Copyright (c) 1998 David Purdue <David.Purdue@computer.org>    */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

main(int argc, char *argv[])
{
        char    **final_path, *path_ptr, *end_dir;
        int     num_colons, i, j, num_dirs, dir_length, found;

        /* Count the number of colons in the path. */
        num_colons = 0;
        path_ptr = index(argv[1], ':');
```

```
        while (path_ptr != NULL)
        {
                num_colons++;
                path_ptr++;
                path_ptr = index(path_ptr, ':');
        }


        /* There will never be more directories out than in. */
        final_path = (char **)calloc(num_colons + 1, sizeof(char *));

        path_ptr = argv[1];
        num_dirs = 0;

        /* Load up the path. */
        while (path_ptr != NULL)
        {
                end_dir = index(path_ptr, ':');

                if (end_dir == NULL)
                {
                        dir_length = strlen(path_ptr);
                }
                else
                {
                        dir_length = end_dir - path_ptr;
                }

                found = 0;

                for (j = 0; j < num_dirs; j++)
                {
                        if (strncmp(path_ptr, final_path[j], dir_length) == 0)
                        {
                                found = 1;
                                break;
                        }
                }

                if (found == 0)
                {
                        final_path[num_dirs] = malloc(dir_length + 1);
                        strncpy(final_path[num_dirs], path_ptr, dir_length);
                        final_path[num_dirs][dir_length] = '\0';
                        num_dirs++;
                }

                if (end_dir == NULL)
                {
                        path_ptr = NULL;
                }
                else
                {
                        end_dir++;
                        path_ptr = end_dir;
                }

        }

        /* Write out the path. */
        for (i = 0; i < (num_dirs - 1); i++)
        {
                printf("%s:", final_path[i]);
        }
        printf("%s\n", final_path[num_dirs - 1]);

        /* Clean & Go. */
        for (i=0; i < num_dirs; i++)
        {
                free(final_path[i]);
        }
        free(final_path);
        exit(0);
} /* main() */
```

❖

CONTINUOUS PS

From: Matthew Dawson <dawson.matthew.ms@bhp.com.au>

Occasionally I need to keep track of what processes are starting up/shutting down on a UNIX system, usually when monitoring or testing new applications that have been installed on a machine. This is especially useful when trying to determine whether applications which spawn child processes are behaving as expected.

I initially tried using plain old ps, but the time taken to type the command and interpret its output caused me to miss certain vital events. So I wrote the following script - contps - to do the job for me. One word of warning; this script was developed under AIX, and as such will require modification to run on some UNIX variants.

```
#!/bin/sh
#     contps (Continuous ps)
#
#     Shows processes as they start/stop.  Use CTRL-C to exit.
#
#     Usage:
#       contps [-i n] [-g string] [-h|-?]
#     where
#       -i n           n is the number of seconds between ps checks
#       -g string      string will be used to grep the ps output
#       -h|-?          Displays usage information
#
#     Copyright (c) 1998 Matthew Dawson <mattd@auug.org.au>

TMPDIR=/tmp                         # Temporary file directory
sleepsecs=1                         # Default ps interval

oldpsfile="$TMPDIR/psold$$.log"     # Previous Processes
newpsfile="$TMPDIR/ps$$.log"        # Current Processes
tempfile="$TMPDIR/psdiff$$.log"     # Temporary File

# Cleanup resources on exit
trap "rm -f $oldpsfile; rm -f $newpsfile; rm -f $tempfile; exit 0" 1 2 3 15

touch $oldpsfile              # Ensures diff will have two files to check.

while [ "$#" -ne "0" ]        # Process shell script options
do
  case "$1" in
    -i)     shift             # Set the number of seconds between ps checks
            sleepsecs="$1"
            shift
            continue;;
    -g)     shift             # Only check for certain processes
            greptext="$1"
            shift
            continue;;
    -h|-?) echo "Usage: contps [-i n] [-g string] [-h|-?]
              -i n      n is the number of seconds between ps checks
              -g string  string will be used to grep the ps output
              -h|-?     Displays usage information"
            exit 0;;
    *)      echo "'$1' is an invalid option."
            shift;;
  esac
done

echo "Added? PID    PPID     User     Command"
echo "====== ===    ====     ====     ======="

while [ 1 ]                   # Continue checking until CTRL-C is pressed
do
  # Gets the details for all running processes (excluding this script's).
  ps -ef  -F"     %P  %p  %U   %a" | grep -v "$$" > $newpsfile

  if [ -n "$greptext" ]       # Only check for processes containing this string.
  then
    grep "$greptext" $newpsfile > $tempfile
    mv $tempfile $newpsfile
  fi

  # Checks for added/deleted processes, and shows whether they were added.
  diff $newpsfile $oldpsfile | tr '<>' 'YN' > $tempfile
  no_lines=`cat $tempfile | wc -l | tr -d ' '`
```

```
  if [ "$no_lines" -ne "0" ] # If the number of processes changed, show them.
  then
    tail -`expr $no_lines - 1` $tempfile
  fi

  mv $newpsfile $oldpsfile
  sleep $sleepsecs
done
```

❖

BIGTAIL

From: Graham Jenkins <Graham.K.Jenkins@corpmail.telstra.com.au>

How many times have you been caught trying to do something like:

```
tail -1000 /var/adm/messages | pg
```

and discovered that the standard version of 'tail' won't give you as many lines as you require?

The attached 'bigtail' script provides a solution. If input is being taken from a file, it uses 'sed' to return the lines which are required. Otherwise, it uses 'nawk' to collect the required lines into a circular buffer as they arrive. Depending on your system, you may need to change 'nawk' to 'awk' and/or change the way in which the value of the 'Lines' variable is fed to it.

```
#!/bin/sh
# bigtail        Enables 'tail -1000' etc. operations on standard input or
#                file.  Graham Jenkins, IBM GSA, January 1998.
#
#                Last revised: 980203

Lines="-10"      # Default

case X"$1" in
  X-[0-9]* ) Lines=$1
             shift ;;
  X-*      ) echo "Usage: `basename $0` [-lines] [filename]" >&2
             exit 2;;
esac

case $# in
  1) Wc=`wc -l < $1` || exit 2
     Start=`expr $Wc + $Lines + 1`
     [ \( $? != 0 \) -a \( $? != 1 \) ] && exit 2
     [ $Start -lt 1 ] && Start=1
     sed -n "$Start,\$ p" $1 || exit 2
     exit 0 ;;

  0) nawk 'BEGIN{ j = 1
                  while ( j == 1 ) {j = getline a[n%Lines]; n = NR}
                }
           END  { if( j < 0 ) exit 1
                  Start=NR-Lines+2
                  if( Start < 1 ) Start = 1
                  for ( j=Start;j<=NR;j++) print a[(j-1)%Lines]
                }' Lines=`expr 1 - $Lines` && exit 0
     echo "Memory overflow! Retry with a smaller number of lines,">&2
     echo "else use: `basename $0` -lines filename"              >&2
     exit 2 ;;

  *) echo "Usage: `basename $0` [-lines] [filename]" >&2
     exit 2 ;;

esac
```

❖

Notification of Change

You can help us! If you have changed your mailing address, phone, title, or any other contact information, please keep us updated. Complete the following information and either fax it to the AUUG Membership Secretary on (02) 9332-4066 or post it to

AUUG Membership Secretary
P.O. Box 366
Kensington, NSW 2033
Australia

(Please allow at least 4 weeks for the change of address to take effect..)

❒ The following changes are for my personal details, member #:_______________

❒ The following changes are for our Institutional Member, primary contact.

❒ The following changes are for our Institutional Member, representative 1.

❒ The following changes are for our Institutional Member, representative 2.

PLEASE PRINT YOUR OLD CONTACT INFORMATION (OR ATTACH A MAILING LABEL):

Name/Contact: _______________
Position/Title: _______________
Company: _______________
Address: _______________
_______________ Postcode _______________
Tel: BH _______________ AH _______________
Fax: BH _______________ AH _______________
email address: _______________

PLEASE PRINT YOUR NEW CONTACT INFORMATION:

Name/Contact: _______________
Position/Title: _______________
Company: _______________
Address: _______________
_______________ Postcode _______________
Tel: BH _______________ AH _______________
Fax: BH _______________ AH _______________
email address: _______________

AUUG Secretariat Use

Date: _______________
Initial: _______________
Date processed: _______________
Membership # _______________

AUUG Inc is the Australian UNIX and Open Systems User Group, providing users with relevant and practical information, services and education through co-operation among users.

AUUG OFFERS SOMETHING FOR YOU!!

Education

Tutorials
Workshops

AUUGN

Technical Newsletter

AUUG's bi-monthly publication, keeping you up to date with the world of UNIX and open systems.

Events.....Events......Events

- Annual Conference & Exhibition
- Overseas Speakers
- Local Conferences
- Roadshows
- Monthly meetings

DISCOUNTS

*to all **AUUG events** and **education**.*

Reciprocal arrangements with overseas affiliates.

*Discounts with various **internet service providers, software, publications** and more...!!*

Connections

- Newsgroup aus.org.auug

Application for Individual or Student Membership

Section A: PERSONAL DETAILS

Surname ____________ First Name ____________
Title: ____________ Position ____________
Organisation ____________
Address ____________
Suburb ____________ State ____________ Postcode ____________
Telephone: Business ____________ Private ____________
Facsimile: ____________ E-mail ____________

Section B: MEMBERSHIP INFORMATION

Please indicate whether you require Student or Individual Membership by ticking the appropriate box.

RENEWAL/NEW INDIVIDUAL MEMBERSHIP

Renewal/New Membership of AUUG ☐ $100.00

RENEWAL/NEW STUDENT MEMBERSHIP

Renewal/New Membership of AUUG ☐ $25.00
(Please complete Section C)

SURCHARGE FOR INTERNATIONAL AIR MAIL ☐ $60.00

Rates valid as at 07/96

Section C: STUDENT MEMBER CERTIFICATION

For those applying for Student Membership, this section is required to be completed by a member of the academic staff.

I hereby certify that the applicant on this form is a full time student and that the following details are correct.

NAME OF STUDENT: ____________
INSTITUTION: ____________
STUDENT NUMBER: ____________
SIGNED: ____________
NAME: ____________
TITLE: ____________
DATE: ____________

Section D: LOCAL CHAPTER PREFERENCE

By default your closest local chapter will receive a percentage of your membership fee in support of local activities. Should you choose to elect another chapter to be the recipient please specify here:

____

Section E: MAILING LISTS

AUUG mailing lists are sometimes made available to vendors. Please indicate whether you wish your name to be included on these lists:

☐ Yes ☐ No

Section F: PAYMENT

Cheques to be made payable to **AUUG Inc**
(Payment in Australian Dollars only)

For all overseas applications, a bank draft drawn on an Australian bank is required. Please do not send purchase orders.

-OR-

☐ Please debit my credit card for A$____________

☐ Bankcard ☐ Visa ☐ Mastercard

Name on Card ____________
Card Number ____________
Expiry Date ____________
Signature ____________

Please mail completed form with payment to:

Reply Paid 66
AUUG Membership Secretary
PO Box 366
KENSINGTON NSW 2033
AUSTRALIA

Or Fax to:

AUUG Inc
(02) 9332-4066

Section G: AGREEMENT

I agree that this membership will be subject to rules and by-laws of AUUG as in force from time to time, and that this membership will run from time of joining/renewal until the end of the calendar or financial year.

Signed: ____________
Date: ____________

AUUG Secretariat Use

Chq: bank ____________ *bsb* ____________
A/C: ____________ *#* ____________
Date: ____________ *$* ____________
Initial: ____________ *Date Processed:* ____________
Membership#: ____________

Application for Institutional Membership

Section A: MEMBER DETAILS

The primary contact holds the full member voting rights and two designated representatives will be given membership rates to AUUG activities including chapter activities. In addition to the primary and two representatives, additional representatives can be included at a rate of $70 each. Please attach a separate sheet with details of all representatives to be included with your membership.

NAME OF ORGANISATION: ____________________

Primary Contact

Surname __________ First Name __________
Title: __________ Position __________
Address ____________________
Suburb __________ State __________ Postcode __________
Telephone: Business __________ Facsimile __________
Email __________ Local Chapter Preference __________

Section B: MEMBERSHIP INFORMATION.

Renewal/New Institutional Membership of AUUG ☐ $350.00

Surcharge for International Air Mail ☐ $120.00

Additional Representatives Number ☐ @ $80.00

Rates valid as at 07/96

Section C: PAYMENT

Cheques to be made payable to **AUUG Inc** *(Payment in Australian Dollars only)*

For all overseas applications, a bank draft drawn on an Australian bank is required. Please do not send purchase orders.

-OR-

☐ Please debit my credit card for A$__________

☐ Bankcard ☐ Visa ☐ Mastercard

Name on Card __________
Card Number __________
Expiry Date __________
Signature __________

Please mail completed form with payment to:

Reply Paid 66
AUUG Membership Secretary
PO Box 366
KENSINGTON NSW 2033

Or Fax to:

AUUG Inc
(02) 9332-4066

Section D: MAILING LISTS

AUUG mailing lists are sometimes made available to vendors. Please indicate whether you wish your name to be included on these lists:

☐ Yes ☐ No

Section E: AGREEMENT

I/We agree that this membership will be subject to rules and by-laws of AUUG as in force from time to time, and that this membership will run from time of joining/renewal until the end of the calendar or financial year.

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

Signed: __________
Title: __________
Date: __________

AUUG Secretariat Use

Chq: bank __________ *bsb* __________
A/C: __________ *#* __________
Date: __________ *$* __________
Initial: __________ *Date Processed:* __________
Membership#: __________

UNIX®AND OPEN SYSTEMS USERS

Membership Application

AUUG Inc Secretariat
PO Box 366, Kensington NSW 2033, Australia
Tel: (02) 9361 5994
Free Call: 1 800 625 655
Fax: (02) 9332 4066
email: auug@auug.org.au
ACN A00 166 36N (incorporated in Victoria)

http://www.auug.org.au