

## Andrew Tanenbaum's Distributed Operating Systems

Like televised murder trials, distributed computing is one of those new facts of life that everyone has an opinion on, but no one really knows what to do about. Especially in the area of operating systems, solutions abound but none has gained the kind of support that DOS, UNIX or MacOS has in the uniprocessor world - the kind of support that gives the industry - and the science that goes on around it - a focus and a direction that might speed the acceptance of the technology.

In his new book, *Distributed Operating Systems*, the veteran Dutch OS guru Andrew Tanenbaum surveys the broken landscape of distributed computing and gives his legion of fans the current lay of the land. Tanenbaum is a familiar figure to OS aficionados from his earlier effort, *Operating Systems: Design and Implementation* which presented the issues of uniprocessor OS design through the prism of his own UNIX clone, Minix. The presentation of Minix was one of the key features of that text. Almost every concept discussed there was explained by close reading of a piece of working code taken straight from the Minix kernel. It was a bottom-up approach, entirely appropriate for undergraduates who know C (and almost nothing else).

*Distributed Operating Systems* takes a much more graduate level, top-down approach. In the first six chapters, Tanenbaum presents the problems that distinguish distributed OSes from uniprocessor OSes and some of the proposed solutions. In the process, he divides the subject into five areas that make up the heart of the book - Communication, Synchronization, Processes, File Systems and Shared Memory.

Communication first gives a little background on network protocol layering, carving out the layers that matter to OS design. It covers ATM in some depth, as that promises to be one of the basic building blocks of the new distributed computing environment. Then it's on to a definition of the Client-Server model. All of that provides the background for the focus of the chapter, Remote Procedure Calls.

Synchronization introduces the topic of interprocess synchronization with one of the most interesting and accessible problems in the field - clock synchronization. It covers Lamport's algorithm in detail, then follows that line of inquiry into serialization, atomicity and concurrency control.

As someone more interested in multiprocessors than multicomputers, the chapter on Processes (as well as Shared Memory) is 'the cream in this canole'. It starts with a straightforward explanation of kernel and user-space threads then launches into a long discussion of process scheduling and processor allocation models. The chapter ends with consideration of two special cases - fault tolerant and real-time distributed systems.

File Systems is one of the smallest chapters. It first hits the big topics: naming, sharing, caching, and replication, then follows with a nice case-study of NFS. The chapter finishes with a survey of current research into scalability, mobile users and fault tolerance.

For me, the chapter on Shared Memory (along with Processes) was the justification for buying this book. It covers switched, bus-based and ring-based systems, caching and degrees of consistency. Fair warning though. Thorough familiarity with uniprocessor memory management is an absolute prerequisite for this chapter.

Each of the last four chapters takes about fifty pages to look in detail at a particular distributed OS - Amoeba, Mach, Chorus and DCE. As Tanenbaum's baby, Amoeba gets first billing and deepest treatment but fans of the others will not feel short-changed either. As someone trying to anticipate the Mach-based OS/2 for the PowerPC, I was particularly interested in that section.

Unlike Tanenbaum's earlier book, *Operating Systems: Design and Implementation*, this one is not chained to a particular implementation. OSDI used Minix exclusively throughout the text which is fine if you're in a course, or have the leisure to bring up Minix on your own, but somewhat less useful if you're trying to understand a non-Minix OS.

Despite it's lack of a Minix analog and the corresponding sample code, the book doesn't skimp on real-world applicability. Throughout chapters 1-6, it takes illustrative examples from many different implementations, from NFS's distributed file system to the BBN Butterfly's shared memory architecture, and discusses them in varying levels of detail in distinct sub-sections. Viewed in its entirety, the vast menagerie of real-world implementations discussed in detail, or mentioned in passing here is breathtaking.

More academic texts often use cold, dry language that makes reading them somewhat akin to licking a piece of steel plate. Tanenbaum, on the other hand, employs a direct, almost conversational, style, liberally spiced with analogies from outside the realm of computer science. He doesn't mince words either when describing some of the messier areas of current technology. He is especially frank in dealing with the odd framing standards of ATM, calling SONET's 810 byte frame *"not something a computer scientist is likely to propose."* He can also display a biting wit as shown in his summary of the current state of the art in distributed deadlock detection: *"Thus we have an active research area in which the model of the problem does not correspond well to reality, the solutions found are generally impractical, the performance analyses given are meaningless, and the proven results are frequently incorrect. To end on a positive note, this is an area that offers great opportunities for improvement."* As nerd-speak goes, it doesn't get any better than this.

This book is an easy and enjoyable read, but it is not for the faint of heart. An undergrad level understanding of uniprocessor operating systems is a must to get

even the smallest benefit. Before reading this, I conducted a brief net.troll for distributed operating systems in comp.arch, comp.parallel.\* and various other newsgroups and mailing lists. This provided valuable perspective, as Tanenbaum touches, at least briefly, on almost all the current research and it's often difficult to tell from a mention in the text whether a particular research branch, like Mach or Linda, is ongoing or dead-ended.

While an OS 'groupie' can get a great deal of benefit from the text alone, the problem sets at the end of each chapter are strictly for hard-core operating system writers. The questions are not 'tricky', but require true familiarity with the material.

If there is one problem with this book, it would be the breadth of material presented. Some subjects that support an entire literature of their own, such as ATM or Mach, are dispensed with in a single sub-section. If you don't have any background in the subject, the chapter-end problem can really take the air out of your sails.

This unnerving breadth is more a function of the state of the art, than of any lack on the author's part. The field is in a state of ferment. This fact is implicitly acknowledged by the author's relentless footnoting. It is a rare page in this book that does not have at least one in-line reference to an outside source such as (see *Agarwal, et al. 1988*). While these references occasionally signal an abrupt end to an interesting line of inquiry, they also make the book an ideal starting point for exploring any branch of the science.

Given the state of the art in distributed operating systems, this is about as good a treatment as you are likely to find. That recommendation comes with just one qualification. Since the field is so active, it is likely to 'narrow' rapidly as various disciplines (protocols, scheduling, memory management, ...) coalesce around standard solutions and discard dead branches of inquiry. Thus, it is more important than usual that this text be updated to reflect current thinking.