The Topaz System: Distributed Multiprocessor Personal Computing

Paul McJones and Andy Hisgen
Digital Equipment Corporation
Systems Research Center
130 Lytton Avenue
Palo Alto, CA 94301

September 28, 1987

1 Introduction

The Topaz distributed operating system, which runs on the Firefly multiprocessor, was built at Digital's Systems Research Center (SRC) in order to support current computing needs and to provide a basis for research into distributed personal computing. In this paper, we give a brief overview of Topaz, and then make some observations based on our use of the system and our plans for the future. These observations fall into two areas: our experience with multiprocessing in a workstation, and our views on distribution and autonomy.

2 Topaz overview

Topaz runs on the Firefly multiprocessor [8]. The Firefly currently couples five MicroVAX II processors to 16 megabytes of RAM via coherent caches. One of the processors has a Q-bus interface, to which are attached controllers for Ethernet, disks, displays, etc.

The Topaz architecture can be viewed as a hybrid of Berkeley's 4.2BSD UNIX[1] [3] and Xerox's Cedar [7]. Topaz borrows the 4.2BSD file system and large-grain process structure, populates these processes (address spaces) with Cedar-like threads of control, and interconnects them with Cedar-like remote procedure call (RPC). (A second implementation of Topaz is layered on UNIX; we use it to run various Topaz servers on large VAX computers.)

One success of Topaz has been its support both for programs using the standard 4.2BSD kernel-call interface and for multithreaded programs using a new Topaz operating system interface. These two kinds of programs can run on the same machine, share files, send each other signals, and start processes of either kind. As a result, we've been able to incorporate many standard UNIX programs (e.g., shells, text editors, and language processors) into our environment, while still being able to explore the benefits of threads (multiple program counters executing in an address space) and RPC. More details about the Topaz operating system, including the reference manual for the file system and process manager, are available in [4].

Topaz itself, and most of the application programs designed at SRC, are written in Modula-2+ [5], which extends Modula-2 [9] with concurrency [1], exception handling, and garbage collection. Topaz RPC is similar to the system described by Birrell and Nelson [2]; it is used not only between machines, but also, with a special transport, between address spaces on the same machine.

3 Multiprocessing

A distinguishing feature of Topaz is its support for concurrent programming within a single application program. Topaz provides the abstraction of threads, which are scheduled concurrently on the Firefly, so programmers can use them to decrease turn-around time by executing independent subcomputations in parallel. Threads are also quite useful in structuring programs that must deal with asynchrony.

While we are pleased with our use of multiprocessing, it has certainly not been a panacea. First of all, five 1-MIPS processors rarely equal one 5-MIPS processor. One of our applications (a utility for copying file trees, somewhat like `rdist(1)`) speeds up by a factor of about 4.7, our parallel implementation of `make(1)` speeds up compilation by a factor of 2 or 3, and our file system and window manager use multiple processors for read-ahead, write-behind, and pipelining. But a single-threaded program runs at the same speed as on a MicroVAX II (or, if it is compute-bound, perhaps 6% faster because extra CPUs are available to run daemons and interrupt routines).

While the benefits of multiprocessing are not always spectacular, we found the costs to be reasonable. The extra VLSI CPUs and caches constitute a modest part of the overall cost of

[1] UNIX is a trademark of AT&T Bell Laboratories

a Firefly (including RAM, controllers, peripherals, power, and packaging). Porting the standard UNIX implementation to run on a symmetrical multiprocessor would have been tricky, but we had the advantage of starting from scratch and being able to use the full power of Modula-2+ (including threads) within the operating system.

To summarize, we advocate that operating systems provide the thread abstraction as a basis for dealing with asynchrony. We also advocate selecting the fastest CPU available, and using multiprocessing mainly when it is worth designing concurrent algorithms to speed up a specific application (on a workstation) or to get more throughput (on a multi-user system).

4 Distribution and Autonomy

Currently, each Firefly has its own disks and local file system. References to remote path names are embedded within symbolic links of the local file system. Accesses to remote files support the usual UNIX semantics, with no caching of remote files. We currently simulate a "global" (i.e., laboratory-wide) name space by following a set of conventions, and running update utilities we have written. This has worked reasonably well, but a distributed file system presenting a uniform name space to a set of machines, such as the ITC [6] and Sprite systems, is clearly a good idea and is something we plan to implement.

We believe that the structure of such a global name space needs to permit individual users and groups of users to have control over their view of the name space. For example, in a software development environment, a programmer or group of programmers may need to view a name space that includes their own experimental versions of a file or of a collection of related files. Similarly, an organization may want to make its own decisions about when to incorporate new releases of software. (IBM Almaden's Quicksilver design has support for customized naming, on a user-specific basis.)

References

[1] A. D. Birrell, J. V. Guttag, J. J. Horning, and R. Levin. Synchronization primitives for a multiprocessor: A formal specification. In *Proceedings of the Eleventh Symposium on Operating System Principles*, New York, 1987. ACM. To appear.

[2] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[3] William N. Joy, Eric Cooper, Robert Fabry, Samuel J. Leffler, Marshall Kirk McKusick, and D. Mosher. 4.2BSD system manual. In *UNIX Programmers Manual, 4.2 Berkeley Software Distribution*, volume 2C. Computer Systems Research Group, University of California at Berkeley, 1983.

[4] Paul R. McJones and Garret F. Swart. Evolving the UNIX system interface to support multithreaded programs. Report 21, Digital Equipment Corporation, Systems Research Center, September 1987.

[5] Paul Rovner. Extending Modula-2 to build large, integrated systems. *IEEE Software*, 3(6):46–57, 1986.

[6] M. Satyanarayanan, John H. Howard, David A. Nichols, Robert N. Sidebotham, Alfred Z. Spector, and Michael J. West. The ITC distributed file system: Principles and design. In *Proceedings of the Tenth ACM Symposium on Operating System Principles*, pages 35–50, New York, December 1985. ACM.

[7] D. C. Swinehart, P. Z. Zellweger, and R. B. Hagmann. The structure of Cedar. In *Proceedings of the ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments*, pages 230–244, New York, June 1985. ACM.

[8] Charles P. Thacker and Lawrence C. Stewart. Firefly: a multiprocessor workstation. In *Proceedings of the Second International Conference on Architectural Suport for Programming Languages and Operating Systems*. ACM and IEEE Computer Society, October 1987.

[9] Klaus Wirth. *Programming in Modula-2*. Springer-Verlag, third edition, 1985.