[illegible]

M.I.T. Laboratory for Computer Science

Request for Comments No. 273
February 11, 1985

Trip Report: Europe, January 25 – February 1, 1985

by Larry W. Allen

My itinerary was: joint ACM/IBM Zurich Research Laboratory Workshop on Operating Systems in Local Area Networks, 3 days; Cambridge University Computer Laboratory, 1 day.

1. Workshop on Operating Systems in Local Area Networks

There were roughly 50 attendees at the workshop, about half of whom were from the U.S. and half from Europe. Liba Svobodova (who chaired the workshop) and the various members of the program committee tried to keep the sessions oriented towards the issues in building distributed systems; despite their best efforts, several of the sessions turned into simply summaries of individual projects. Nevertheless, I learned a number of interesting things.

1.1. Remote File Access

There was a fair amount of discussion on providing remote file access to systems in a local area network. The main distinction raised seemed to be between systems which provide true, block-at-a-time (or byte-at-a-time) access to remote files across the network, and systems which use a local disk as a cache of recently-accessed files and which transfer entire files across the network to the local disk when necessary. Michael Schroeder of the DEC Systems Research Center discussed the Cedar file server at Xerox PARC, which uses local disk caching; M. Satyanarayanan of CMU's Information Technology Center (their equivalent of Project Athena) discussed a similar scheme to be used there. Remote file-access schemes were discussed by Lindsay Marshall of Newcastle University, among others.

The local-cache supporters claimed that most applications read an entire file sequentially from beginning to end anyway; and that remote file access schemes will be slow because of server overloading when serving many users. The remote-access supporters mainly argued that local disks are undesirable due to the cost, noise, heat, and high maintenance requirements of disk drives. Concern was also raised over the handling of large, database-like files (for example, the dictionary on UNIX systems). Such files are often sparsely accessed. The consensus was that some sort of special-casing was required for database files.

A clear consensus did not emerge from the workshop on this topic. The DEC SRC people will soon be designing a file service for their new system; it will be interesting to see what design they settle on.

1.2. Language Support for Communication

My impression here is that the general trend is strongly towards the use of Remote Procedure Calls and towards lightweight concurrency. Although some message-passing systems were described, there seemed to be nearly universal support for RPC. The arguments in favor of RPC boiled down to the fact that RPC seems to greatly simplify the task of building distributed applications. Many things that have to be done explicitly in message-passing systems (specifying message formats, encoding and decoding data objects for transmission, handling timeouts and retransmission) are done for you in an RPC-based system.

Most people seemed to be extending existing languages (CLU, Modula-2) to support these features. It's interesting to note that existing languages still apparently don't have the kind of communications support people want. This is emphatically true of Ada. There were two reports on attempts to implement distributed systems in Ada; both found it necessary to add extensions to the language. In particular, Ada's tasking structure (which is quite heavyweight) and the rendezvous mechanism were strongly criticized.

There was surprisingly little discussion of other programming environment features needed for building distributed systems, such as debuggers or performance monitoring tools. It appears that interest in this topic has died down somewhat, although I don't believe any of the problems have been satisfactorily solved.

1.3. Upcalls

I spoke in the session on "Operating System Support for Communications". My talk concentrated on the multi-task module and upcall ideas as used in Swift. The talk generated a modicum of interest. Later, I discussed upcalls with Roy Levin of DEC SRC, particularly concentrating on how upcalls could be used in the system being built there. We reached a general agreement that most client programs (and most client programmers) don't want to have to think about asynchrony, so in general upcalls shouldn't propagate all the way up to the client level. Inside a subsystem (for example, a window system), and between lower-level subsystems, upcalls can be a very useful programming technique.

1.4. Conclusion from the Workshop

The overall conclusion I reached from the workshop was that building a distributed system today is basically an engineering problem, not a research topic. Most of the serious problems – security, atomicity, recoverability – have fairly well-understood and accepted solutions. Many of the participants are now designing or building their second major distributed system, and trying to "do it right this time", but few seem to be breaking new ground in this area.

Several participants pointed out that there is something of a lack of good distributed applications. I suspect this lack will be rectified in the next couple of years, as some of the new systems now being built become operational.

2. Cambridge University Computer Laboratory

My host at Cambridge was Andrew Herbert, who spent last summer at MIT working with us on Swift. I spent a good deal of time talking with Graham Hamilton and Dan Croft, who are implementing the Mayflower operating system. Cambridge will soon acquire 12 microVAX processors, to be used in a new processor bank, and some number of Xerox Dandelion (Star) workstations, to be used as user interface machines. They are very interested in getting our Remote Virtual Disk software for UNIX, as the microVAXes will run UNIX at least initially.

The Mayflower system is currently running; I was given a demonstration of the new version of the Resource Manager (as described by Dan Croft at the last SOSP) running under Mayflower. Current work is centering around the construction of a source-language debugger, and integration of Mayflower and the Resource Manager system.

Current projects at Cambridge include Mayflower; a high-speed (100 Mbit/sec) version of the slotted ring; and a project to use the high-speed ring for digital telephony. There is a great deal of discussion about how to best use the new hardware which will soon be arriving; in particular, there is some possibility that Mayflower may be ported over to the microVAX.

I spoke on Swift to a group of about 25 people from the Computer Laboratory. There was quite a bit of interest in our use of upcalls, and also in garbage collection issues. Several people shared my interest in exploring the possibilty of using some sort of hardware-assisted reference-counting scheme instead of a mark-sweep garbage collector; the advantages being that storage can be reclaimed incrementally (a major factor in a large-address-space system), and the required amount of hardware assistance is quite small. We discussed the differences in approach of Swift and Mayflower; many of the differences arose from the desire to use Mayflower in the environment of a processor bank.

Andrew Herbert will be leaving the Laboratory in June to take over a position as Chief Architect of a distributed systems project, funded by Project Alvey (Britain's Fifth Generation computer project). It's not yet clear what effects his departure will have on the Mayflower or microVAX projects.