Research Proposal
Thread Tunnelling in Vertically-Structured Systems
Stephen Kell
Draft, 27 May 2005

Motivation

Vertically structured operating systems such as Nemesis [1] and Exokernel [2] offer benefits of extensibility and resource accounting and are suited to providing guarantees to soft real-time applications (e.g. video streaming). This is achieved by multiplexing resources at the lowest possible level, and having each process perform all higher-level operations under its own guarantees.

Unfortunately, sharing state between processes presents difficulties under this model. Shared memory is available, but is often inadequate: to maintain correctness, security or fairness it may not be desirable to share data structures directly. Traditionally, a server process is used, maintaining the shared state within its own protection domain and allowing controlled access using inter-process communication (IPC) mechanisms. Since these servers are independently scheduled, the resources used (including CPU time) cannot be accounted to the clients which use them.

At the root of this problem is the convention that scheduling and resource consumption are tied to protection, privilege and resource ownership. The technique of thread tunnelling orthogonalises these by permitting an independently-scheduled thread to move between protection domains, executing the host domain's code holding the host domain's privileges and resources. (This scheduling paradigm is known as 'path-based scheduling', since it effectively schedules the processing path of a data item rather than the individual tasks along that path.) As an IPC mechanism, thread tunnelling offers very low latency, since transfer of data and control occur together and immediately.

Existing Work

Early implementations of thread tunnelling were done hardware on the CAP computer [3] and in software by the TAOS system [4]. Nemesis rejected thread tunnelling since it necessitates kernel-implemented threads, which were considered harmful to the ideal of maximising application control over internal resource allocations. Since the design of Nemesis, kernel threading has become more widespread and the original problems open to be re-assessment. The Expert [5] NEOS uses thread tunnelling but is substantially based on Nemesis, while Spring [6] and a modified Mach [7] also use the technique.

Proposal

Integrating tunnelling into a general-purpose operating system presents considerable difficulties, and so far the most satisfactory implementations have been special-purpose systems such as Expert. The research proposed is to investigate one or more ways in which tunnelling may be added to existing general-purpose systems in order to improve fairness and/or increase IPC performance.

Implementation

The obvious approach is to add tunnelling directly to Nemesis, similarly to Expert but for the general use case. However, the overlap with Expert is likely to be substantial, and with Nemesis no longer the subject of research as a general-purpose OS, it might be preferable instead to target a more active or widely-used platform.

Adding tunnelling to Linux (or specifically a variant supporting QoS guarantees, such as QLinux [8]) would be possible only in limited cases. Tunnelling into the kernel across the system call interface is logically plausible, but likely to be difficult to implement, since such a large change to a large monolithic system might have extremely complex consequences for the rest of the system.

A more straightforward (but limited) Linux-based option would be adding kernel support for a tunnelling-based user-level IPC scheme which could be used in place of pipes or Unix domain sockets. Processes would export a named, typed entry point (reminiscent of a 'gate' in Multics [9] or a 'door' in Spring) which a thread could tunnel through using a special system call. A user-level software stack (such as an X11 server and client libraries, user-level filesystem or user-level networking implementation)

could be modified to use this mechanism and compared with conventional implementations in terms of both QoS crosstalk and IPC performance.

A final option lies with the Xen virtual machine monitor project [10]. Xen implements physical device drivers in individual protection domains with restricted privilege [11]. These are schedulable entities and may be shared among guest OSes. A unified interface (analogous to an IPC interface) is used to access the driver. It would be desirable for CPU time spent by the drivers to be accounted to the client OS, which could be achieved using tunnelling.

Evaluation

The evaluation of the resulting system would consist of testing IPC latency and QoS crosstalk in comparison with the original system. IPC latency can be measured in observed performance of IPC-intensive applications (such as a windowing system), while QoS crosstalk might be measured by performance of a continuous-media application on a loaded system or in the presence of background tasks with bursty resource usage.

A target for the first year of research would be to have selected a suitable target system, determined what changes are required to support tunnelling and implemented enough of these to produce a proof-of-concept tunnelling system, most likely as a special case with many limitations remaining.

Other Approaches

The extents of the gain to be had from adding tunnelling is open to question. Although Nemesis uses shared servers, it was designed to keep their use to a minimum. It may be that the potential for improved performance or reduced crosstalk is insignificant, or outweighed by the constraints placed on intra-application scheduling under kernel-based threading.

With the addition of tunnelling, microkernel systems such as Spring and vertical systems such as Nemesis appear more similar. When the shared servers of a microkernel become passive entities open to tunnelling, they can be seen as library code 'instantiated' within an autonomous (but passive) domain. This contrasts with conventional shared libraries, which have no autonomy but exist entirely within host domains, with a duplicated set of internal state for every 'client'.

An interesting alternative question might therefore be to investigate of trade-offs along this spectrum of tunnelling-based systems: a library-based system will consume more resources, but have fewer problems of contention and hence be able to provide better QoS guarantees.

References

[1] I.M. Leslie et al, "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications", IEEE Journal on Selected Areas in Communications, 1996.

[2] D.R. Engler et al, "Exokernel: An Operating System Architecture for Application-Level Resource Management", Proceedings of the ACM Symposium on Operating System Principles, 1995.

[3] M.V. Wilkes and R.M. Needham, "The Cambridge CAP Computer and its Operating System", North Holland, 1979.

[4] B.N. Bershad et al, "Lightweight Remote Procedure Call", ACM Transactions on Computer Systems, vol. 8 no. 1 pp. 37-55, February 1990.

[5] A. Donnelly, "Resource Control in Network Elements", University of Cambridge Computer Laboratory Technical Report no. 534, April 2002.

[6] G. Hamilton and P. Kougiouris, "The Spring Nucleus: A Microkernel for Objects", Proceedings of the 1993 Summer USENIX Conference, 1993.

[7] B. Ford and J. Lepreau, "Evolving Mach 3.0 to a Migrating Thread Model", Proceedings of the 1994 Winter USENIX Conference, January 1994.

[8] P. Goyal et al, "QLinux: A QoS Enhanced Kernel for Multimedia Computing", `http://lass.cs.umass.edu/software/qlinux/`.

[9] E.I. Organick, "The Multics System: An Examination of Its Structure", MIT Press, 1972.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt and A. Warfield, "Xen and the Art of Virtualization", Proceedings of the ACM Symposium on Operating Systems Principles, 2003.

[11] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield and M. Williams, "Safe Hardware Access with the Xen Virtual Machine Monitor", Proceedings of the 1st OASIS Workshop, October 2004.