Network performance in Arena

K.R. Mayes, J. Bridgland, S. Quick and A. Nisbet

University of Manchester, Oxford Road, Manchester, UK.
email: ken@cs.man.ac.uk bridglaj@cs.man.ac.uk
http://www.cs.man.ac.uk/cnc/arena

1 Introduction

Arena is a customisable distributed operating system. Customisable systems seek to provide application-specific resource management. A major consideration in such systems, particularly where customisation is supported at user-level, is the issue of flexibility versus efficiency. This issue is seen in distributed systems, where the performance of handling network traffic is of prime importance. This paper describes the mechanisms used in the Arena customisable operating system[1] (Mayes et al, 1994) to support flexible and efficient network handling on a distributed store multicomputer. In order to gain some relative measure of the costs of flexibility in the Arena event handling mechanism, Arena network data transfer performance has been compared with that of a microkernel.

2 Basic Structure of Arena

Resource management *policy* in Arena is implemented at user-level. Resource managers are represented by libraries of manager objects that are linked to the application. Access to the *mechanisms* of the hardware is provided by a layer of hardware-dependent code which provides a low-level abstraction of processor and store hardware (Quick, 1996). These abstractions form an interface which can be implemented native on many processors for portability. The initial target architecture for the distributed implementation of Arena is the European Declarative System prototype machine (EDSP). This is a distributed store multicomputer with an internal network connecting the processing nodes, each of which has two Sparc processors sharing 64 Mb of nodal store (Ward and Townsend, 1990).

Arena provides an optimised transfer, where the application buffer is used directly by the network driver which executes in supervisor mode. A hardware event may cause an upcall to be made into user-level, to allow deferred processing of the event by an event handler thread. The policy component of *how* to handle the event is determined by the application-specific event handler thread code. Furthermore, such event handler threads are explicitly scheduled by the user-level Process Manager (PM) scheduler code, selected for the application, so that the

[1]This work is supported by EPSRC grants GR/J 84045, 93315512 and 91309499.

application can also make policy decisions about *when* to service deferred events. Allowing applications to tailor the responses to events may introduce inefficiency due to the context switch to the handler thread.

3 Performance Results and Discussion

Performance figures for transferring data between two nodes of the EDSP machine were measured for Arena and for a microkernel (Ward and Townsend, 1990). The values for the microkernel were obtained using an *RPCSend() RPCReply()* pair. Arena performance was faster than that of the EDSP microkernel. With a transfer of 4Kbytes of data the figure for Arena was 527μs versus 1427μs for the microkernel, and with a transfer of 128 bytes the Arena result was 281μs versus 792μs. Since a single packet contains 128 bytes of data, the processing overhead will be relatively large for that transfer size.

In both systems, this time represents a user-level to user-level transfer of data. Both systems use user-level message buffers in the transfer with no intermediate copying. Whereas Arena is executed only on one of the processors of each node, the microkernel partitions processing across both processors, running network code exclusively on one processor. The main point to note is that the cost of running the Arena user-level event handler threads does not represent a significant overhead, compared to the figures for the microkernel. It may thus be argued that the flexibility afforded by the Arena upcall mechanism does not carry too high a cost.

4 Comparisons with Other Work

The Streams approach of Unix for handling character I/O allows deferred servicing of messages at kernel-level. In the Arena approach, deferred processing is by specific user-level handler threads which are explicitly scheduled by the application PM, so that deferred servicing occurs in a predictable fashion. Raven maps device registers into a user address space so they are directly accessible to user-level tasks responsible for event handling (Ritchie and Neufeld, 1993). The Arena approach seeks to combine security, flexibility and efficiency in networking by executing device driver code at supervisor-level, allowing policy to be implemented as user-level handler threads, and providing a means of avoiding copying.

5 References

Mayes, K., S. Quick, J. Bridgland and A. Nisbet (1994) Language- and application-oriented resource management for parallel architectures. *Proceedings of the 6th ACM SIGOPS European Workshop*, September, 172-177.

Quick, S. (1996) A generalised hardware interface for operating systems. PhD Thesis, University of Manchester (In Preparation).

Ritchie, D.S. and G.W. Neufeld (1993) User level IPC and device management in the Raven kernel. *Usenix Association Proc. Symp. on Microkernels and other Kernel Architectures* (Sept), 111-125.
Ward, M. and P. Townsend (1990) EDS hardware architecture. *Lecture Notes in Computer Science* 457, 816-827.