Functional divisions in the Piglet multiprocessor operating system

Steve Muir, Jonathan Smith[1]
{sjmuir,jms}@dsl.cis.upenn.edu
Distributed Systems Laboratory
University of Pennsylvania

Abstract

As multiprocessor computer systems become more commonplace, and peripherals are built with on-board CPUs, we believe that new operating system models are required to make the most efficient use of such systems. At the same time, the role of computers is changing from a computational device to a communications tool, thus emphasising the ability to efficiently support multimedia communication rather than computation alone.

These changes prompted the development of Piglet, an asymmetric multiprocessor operating system. Piglet partitions processors into functional groups in order to better utilise multiple processors. We describe the implementation of Piglet and show how it can provide efficient multiplexing of shared resources.

1 Introduction

The traditional approach to operating system design for multiprocessor computers has been symmetric multiprocessing (SMP), due to the potential for increased job throughput. Several important trends suggest rethinking the structure of operating systems, and in particular the preference for symmetric multiprocessing. These are:

1. the changing role of computers, from multiuser computing servers to personal workstations used primarily for multimedia communication.
2. the increasing presence of processors on peripheral cards.
3. the increasing need for systems which can control their delivered quality-of-service (QoS).

The last few years have seen a transition in the role of computers, away from shared servers used for simulation and management of persistent storage and towards desktop machines used primarily for communication with other people. This new computational model is dissimilar from what is usually presumed in multiprocessor system design and benchmarking.

The presence of processors on peripheral cards is an exciting phenomenon which is increasingly common e.g. the I_2O architecture [1] and the *Myrinet* network [2]. There have been some studies on the use of such facilities in network adaptors [3, 4, 5], and their use will increase. We believe that the role these processors can play in supporting applications should be examined. Prototype systems can be built to investigate these 'smart' devices by using one or more CPUs in a multiprocessor as logical I/O processors.

Finally, the role of the computer includes processing and delivery of continuous multimedia data, and this, as well as security concerns such as denial-of-service threats, drives the need for controllable QoS. Since traditional SMP systems are generally extended versions of general purpose operating systems, they share with these systems the difficulty of robust scheduling for QoS.

2 Previous Work

Several efforts have proposed new, radically different system structures as ways of supporting the requirements of new computing models and applications, in particular increased communication and guaranteed QoS.

The central tenet of the *Nemesis* system [6] is that resource multiplexing should be done once at the lowest possible level in the system. The resulting system structure is a single-address space operating system with a time-division multiplexing algorithm operating at the lowest system level; even the majority of interrupt service is done under scheduler control, and accounted to a process.

[1]This research was supported by NSF grant CDS-97-03220 and DARPA contracts #N66001-96-C-852, #MDA972-95-1-0013, and #DABT63-95-C-0073. Additional support was provided by the AT&T Foundation, and the Hewlett-Packard and Intel Corporations.

The MIT *Exokernel* [7], while sharing many goals and ideas with Nemesis, focuses more on the extensibility inherent in its structure, with the goal of constructing application-specific O.S. structures.

Both of these systems could be classified as 'vertical' operating systems, since they replace the traditional 'horizontal' boundary between the O.S. kernel and applications with vertical boundaries between each application domain (which logically extend all the way down to a very thin interface above the hardware).

As stated earlier, a number of network interfaces have been built with processors on-board, such as the *Myrinet* [2] and *Osiris* [3] devices. The U-Net [4] project used the I/O processor on the Myrinet interface to provide applications with direct access to the network interface, as did Druschel's *Application Device Channels* for the Osiris interface. The *SPINE* project [5] at the University of Washington is investigating ways of moving O.S. and application functions onto the Myrinet device's processor in a safe and controlled manner.

I_2O [1] is a more general architecture for the coupling of processors with I/O devices, not just applicable to network interfaces. It proposes the use of a general-purpose RISC processor to provide 'intelligence' to peripherals.

3 The Piglet Operating System

While systems such as Nemesis and the Exokernel have managed to successfully address many of the problems with conventional O.S.'s described earlier, they require comprehensive restructuring by application designers and system users. We believe this is the primary reason why they have not been readily adopted.

Intelligent devices have also failed to take off as rapidly as might have been expected, the reasons for which undoubtedly include cost and complexity. However, a perhaps more fundamental problem with such devices is their relatively long development cycle compared to the extremely rapid increase in power of standard CPUs. Computational power of the latter is growing at such a rate that the amount of processing which can be usefully off-loaded to a much less powerful I/O processor is rapidly diminishing.

We propose a new operating system structure which pulls together ideas from both vertical operating systems and intelligent devices, providing many of their benefits without requiring the extensive system restructuring of the former or specialised hardware of the latter.

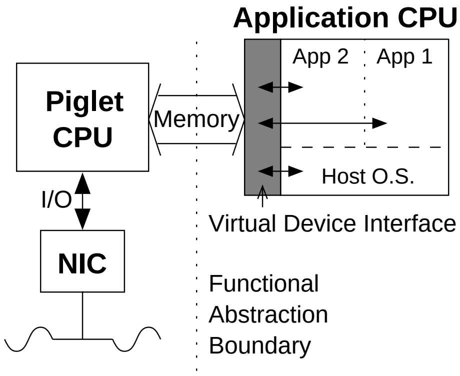

Figure 1: Functional Division in Piglet

Our system, Piglet, has been implemented on multiprocessor Intel PCs, and provides controlled resource sharing to standard applications running on a commodity O.S. (Linux). It also allows functionality to be partitioned between the host system, upon which the host O.S. and applications execute, and Piglet's *Lightweight Device Kernel*, which provides *Virtual Device Interfaces* to the host system to extend the functionality of commodity devices (see Figure 1).

3.1 Functional Division

The fundamental principle upon which Piglet is based is that of *functional division*—dividing application functionality between multiple processing elements.

Functional division is performed at a level of abstraction, for example the definition of a virtual device model. Piglet utilises one or more system CPUs to present *Virtual Device Interfaces* directly to user-space applications (as shown in Figure 1). Applications communicate directly with Piglet via shared regions of memory, thus providing a high-bandwidth, low-latency communication channel.

This functional partitioning has numerous benefits, chief among them being the performance increase due to concurrency between processing elements, and distribution of work-load across those elements. Secondary benefits include removal of device-specific code from the host O.S., and the ability to transparently add functionality below these interfaces.

As an example of the latter, TCP/IP checksumming functions could be moved from the host protocol stack into the Piglet LDK. As devices which provide these functions in hardware become more common, Piglet can hand-off those computations without changing its interfaces. This idea is further discussed in relation to the *video NIC* interface described in Section 3.2.

While direct device access has long been acknowledged as an important method for providing application-specific resource management, it has previously only been possible with custom 'intelligent' hardware. The lightweight device kernel provides protection and multiplexing of the physical device resources in order to provide direct user-space interfaces to standard 'dumb' devices.

3.2 An Example Virtual Device Interface: the Video NIC

An example of a virtual device interface provided by Piglet is the *Video NIC* described in [8]. Piglet provides applications with an interface through which encoded video frames received from the network are decoded by the lightweight device kernel, and presented to the application using a shared-memory interface.

The application gains access to this service by first opening a standard UDP network socket, then using a socket option to indicate to Piglet that it requires Piglet to perform MPEG decoding on that socket. Control packets sent by the remote peer are passed to the application unmodified by Piglet, while encoded video frames are decoded by Piglet. Piglet then sends a UDP control packet to the application specifying the location of the memory buffer (mapped read-only in the application's address space) containing the decoded frame. Once the application has finished displaying the frame it notifies Piglet using a system call so that the buffer may be re-used.

In the current implementation MPEG-1 encoded video frames are received from the network using a simple flow-control protocol layered on top of UDP. Piglet handles all the details of this protocol internally e.g. the sending of acknowledgements to the sender, so the application need not be aware of the details. After decoding, the frames are presented to the application using the standard (Y, U, V) representation.

Since the application need not be aware of the implementation details of the virtual device interface, the MPEG-1/UDP-based protocol can be replaced with other alternatives without changing the interface (assuming a reasonable interface was chosen initially). For example, both motion JPEG and H.261 encodings of video frames can be readily decoded to the same (Y, U, V) representation as MPEG encoded video. Hence the appropriate protocol can be chosen depending on whatever policy the system administrator specifies.

3.2.1 Video NIC Performance

Preliminary experiments (described in [8]) performed using our video NIC implementation show that the concurrency due to decoding in parallel with rendering produces roughly a 10% increase in video framerate.

We believe that this increase is only moderate due to the uneven division of computation between Piglet and the host CPU. However, this also leads us to expect that the utilisation of the host CPU should be much lower[2], thus freeing up the host system for other tasks.

We intend to investigate the decoding process further to determine whether different divisions of labour are possible, and what their benefits in terms of throughput and utilisation are.

3.3 Current Implementation

Piglet is currently implemented on dual-processor Intel Pentium and Pentium Pro PCs, as an extension to the Linux 2.0.30 kernel. Our current research is focused on Piglet's applicability to high-performance networking, so our preliminary support is for a virtual network device interface. We have implemented a Piglet device driver for the 3Com 3c905 network interface card as a simple set of modifications to the Linux driver.

In order to support efficient multiplexing of network resources across multiple applications (including the Linux kernel, which is not handled any differently than user-space applications) Piglet's virtual network interface is presented using an abstraction called a *frameset*.

Each frameset contains control information, plus receive and transmit queues. Applications send data by writing *frames* into their frameset's transmit queue, and Piglet polls the framesets to dequeue frames for sending via physical network interfaces ('NIF's).

In order to provide some QoS guarantees, Piglet implements Virtual Clock [9] to schedule the movement of network packets from framesets to NIFs. Each queue has associated with it two parameters, the period and the limit, which control the rate and the burstiness at which data may be sent from that queue. In addition, applications which do not use this mechanism, for whatever reason, are partitioned into a separate class, the *available bit-rate (ABR)* class. Framesets in this class may only send packets if no virtual-clock controlled frameset has packets

[2]unfortunately data to confirm this is not currently available

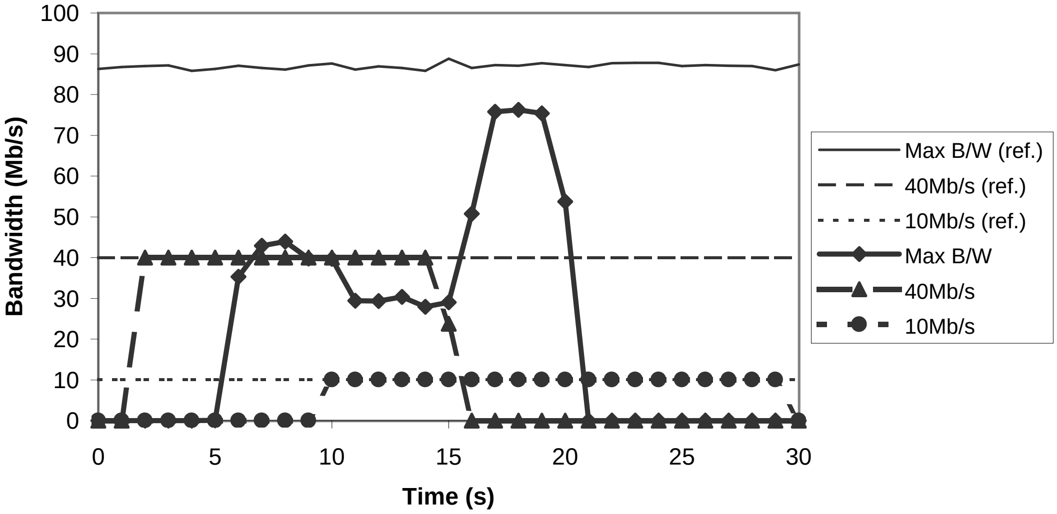

Figure 2: Per-channel bandwidth as a function of time

outstanding.

4 A Resource Multiplexing Experiment using Piglet

As a demonstration of the viability of the Piglet architecture, we considered its applicability to the problem of resource multiplexing in a host O.S. which was not designed to handle application-specific resource requirements.

The specific problem we address is that of multiplexing a single network interface between multiple uncooperative applications, each of which may or may not have specific requirements. In this example the resource we wish to partition between these applications is network bandwidth.

4.1 Experimental Description

The experimental setup consists of two PCs connected to an AsanteFast 100Mb/s Ethernet hub by 3Com 3c905 network interface cards. The sender is a 200MHz dual-processor Pentium Pro PC, running RedHat Linux 5.0 with the Piglet kernel replacing the standard Linux kernel. The receiver is a 200MHz uniprocessor Pentium Pro PC, running RedHat Linux 4.2 with the Linux kernel 2.0.31. Both machines are idle apart from the test applications, and the test network has no other traffic.

The experiment consists of three applications all trying to send a large amount of data from the sender to the receiver. The results are plotted in Figure 2. Each application has different bandwidth requirements:

1. **A** - an unconstrained sender which uses as much bandwidth as is available.
2. **B** - a sender constrained to run at 40Mb/s [3].
3. **C** - a sender constrained to run at 10Mb/s.

The application used to send the data is the standard ttcp utility augmented with an option to set the Virtual Clock parameters (period and time). These parameters are passed to the Linux TCP/IP stack by `setsockopt()` system calls, where they are then passed to Piglet to create application-specific framesets with those parameters. This is the only modification made to the application or to the Linux networking code[4].

Each of applications **A**, **B**, and **C** start and stop sending their data at different times, and the per-application bandwidth is measured every second and plotted in Figure 2 as the three heavy lines. The three thinner lines show reference bandwidth measurements for each sender with no competing applications over a 30 second period.

- After 1s, **B** starts sending at 40Mb/s[5].

[3] Here and in our experimental results we use the convention that 1Mb/s = 10^6 bits per second

[4] We are also investigating the setting of per-flow Virtual Clock parameters using simple rules e.g. all traffic sent to host 158.130.6.140 is constrained to a maximum of 10Mb/s, thus eliminating the need for any application modification at all.

[5] ttcp actually tries to send as fast as possible but Piglet constrains packet transmission to 40Mb/s

- After 5s, **A** starts sending as fast as possible. Piglet's guarantee of 40Mb/s to **B** limits **A** to approximately 40Mb/s also.
- After 9s, **C** starts sending at 10Mb/s, causing **A**'s bandwidth to decrease by approximately 10Mb/s.
- After ≈15s, **B** stops sending, **A**'s bandwidth thus increases to approximately 10Mb/s below the absolute limit.
- After ≈20s, **A** stops sending.
- After ≈29s, **C** stops sending.

4.2 Experimental Results

We see from the graph that Piglet's queue scheduling mechanism provides controlled multiplexing of the shared network resource, despite the fact that the applications are not cooperating to share the resource, and neither they nor the host O.S. (Linux) are aware of the constraints imposed upon them.

The applications which have specific bandwidth requirements receive exactly that amount of bandwidth, even when an application with no specified constraint is competing for the same resource. This ability to add resource management to a standard host O.S. is one of the key strengths of Piglet.

5 Future Work

Future development of Piglet will proceed in several directions, including enhancements to the lightweight device kernel, extensions to the existing video NIC and Virtual Clock services, and the addition of new services.

We intend to extend the LDK with capabilities for extension by user applications in a manner somewhat similar to the SPINE project. This allows applications to install their own services into the LDK and thus obtain the benefits of concurrency and direct access to the network device. We also wish to provide direct application access to framesets, permitting the implementation of user-space protocol stacks [10].

As described in Section 3.2.1 we intend to further investigate the MPEG decoding algorithm to determine if alternative divisions of computation are possible. In order to apply the Virtual Clock mechanisms to traffic sent by applications which are not Piglet-aware, we intend to use packet filtering techniques to partition the single frameset used by the host O.S. for these applications into multiple flows whose parameters are specified by simple rules.

Finally, we believe that the most interesting area of further research is in the provision of services by Piglet. As an example, Piglet could be used to manage multiple line interfaces on a network element, providing various services such as fire-walling, shielding from denial of service attacks, and QoS routing. We are investigating the use of Piglet to support programmable network infrastructures, such as Active Networks [11], and in particular the provision of an efficient interpreter for the PLAN language using runtime specialisation [12].

6 Conclusions

Our experiments with Piglet demonstrate two key advantages which arise from functional division. Firstly, we showed how the Virtual Clock queue management scheme could be embedded in Piglet to provide controlled resource management to applications even if the host operating system does not. Secondly, the video NIC interface serves as an example of a virtual device interface, providing a service to applications which increases performance (higher throughput and lower host system utilisation) and extends the abstractions usually provided by the O.S.

We therefore believe that Piglet can provide many of the benefits of vertical operating systems without requiring the same extensive system restructuring. Similarly, Piglet can provide the much of the same functionality as intelligent devices without being limited to or constrained by specialised hardware. However, we do not claim that Piglet obviates the need for research in those areas, but rather that it is complementary to both, providing a more familiar prototyping and development environment for the investigation of ideas.

Finally, let us address two frequent concerns: that a static division of CPUs may not be appropriate for all workloads, and that an SMP system with multithreaded libraries can provide the same benefits. With regard to the former we have designed a scheme to switch between Piglet and SMP on an as-needed basis; hooks in the host O.S. support Piglet's virtual interfaces while running in SMP mode. The effect is a hybrid system which we believe should perform better across environments than either a simple SMP system or a statically-partitioned Piglet. As for the latter, we believe that the complexity and overhead of supporting direct device access to user-level threads is much greater than in Piglet, where the lightweight device kernel runs with full O.S. privileges.

References

[1] Intelligent I/O Special Interest Group; *"About I_2O Technology"*, **http://www.i2osig.org/Architecture/**

[2] Myricom, Inc.; *"Myrinet: A Brief, Technical Overview"*, **http://www.myri.com/myrinet/overview.html**

[3] Peter Druschel, Larry L. Peterson, and Bruce S. Davie; *"Experiences with a High-Speed Network Adapter: A Software Perspective"*, Computer Communication Review, Volume 24, Number 4 (October 1994), pp.2–13.

[4] Anindya Basu et al.; *"U-Net: A User-Level Network Interface for Parallel and Distributed Computing"*, Proceedings of the 15th ACM Symposium on Operating System Principles (December 1995).

[5] Marc Fiuczynski et al.; *"On Using Intelligent Network Interface Cards to Support Multimedia Applications"*, 8th International Workshop on Network and Operating System Support for Digital Audio Video, Cambridge, England (July 1998), pp.95–98.

[6] Ian Leslie et al.; *"The Design and Implementation of an Operating System to Support Distributed Multimedia Applications"*, IEEE Journal on Selected Areas in Communications, Volume 14, Number 7 (September 1996), pp.1280–1297.

[7] Dawson R. Engler, M. Frans Kaashoek and James O'Toole, Jr.; *"Exokernel: An Operating System Architecture for Application-Level Resource Management"*, Proceedings of the 15th ACM Symposium on Operating Systems Principles (December 1995).

[8] Steve Muir and Jonathan Smith; *"Supporting Continuous Media in the Piglet OS"*, 8th International Workshop on Network and Operating System Support for Digital Audio Video, Cambridge, England (July 1998), pp.99–102.

[9] Lixia Zhang; *"Virtual clock: A new traffic control algorithm for packet switching networks"*, ACM Transactions on Computer Systems, Volume 9, Number 2 (May 1991), pp.101–124.

[10] Aled Edwards and Steve Muir; *"Experiences Implementing a High-Performance TCP in User-Space"*, Computer Communication Review, Volume 25, Number 4 (October 1995), pp.196–205.

[11] Jonathan Smith et al.; *"SwitchWare: Towards a 21st Century Network Infrastructure"*, **http://www.cis.upenn.edu/ switchware/papers/sware.ps**

[12] Scott Thibault et al.; *"Safe and Efficient Active Network Programming"*, **http://www.irisa.fr/compose/talks/plan.pdf**