Supporting Continuous Media in the Piglet OS

Steve Muir, Jonathan Smith[1]

Distributed Systems Laboratory, University of Pennsylvania

{sjmuir,jms}@dsl.cis.upenn.edu

Abstract

Specializing operating system (OS) functions on a per-processor basis can provide efficient support for continuous media, providing better management of resources as well as a model with which parallelism can be applied to data streams. The Piglet OS uses this model.

We describe how Piglet can provide many of the benefits of other OS proposals such as vertically-structured operating systems without the need to change the system model seen by end-users. Our implementation of Piglet provides guaranteed network bandwidth using the Virtual Clock algorithm, and supports parallel (pipelined) MPEG decoding using a virtual device interface.

1 Introduction

Efficiently supporting continuous media places new requirements upon networks and operating systems. Firstly, resources must be managed so that the demands of multiple independent applications can be accommodated. Secondly, the complex encodings of many forms of continuous media require extensive computational power to decode them in a timely manner.

Unfortunately, traditional OS structures cannot meet these requirements due to the assumptions inherent in their design. Their abstractions are too general, designed to present application writers with high-level interfaces to devices e.g. Berkeley sockets. Unfortunately this abstraction makes it difficult for applications to make best use of the underlying hardware.

Traditional operating systems also make it a considerable challenge to the programmer to obtain maximum benefit from parallelism available in the system hardware. For example, writing an MPEG player which makes efficient use of dual processors in an SMP configuration is decidedly non-trivial.

[1] This research was supported by DARPA under 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.

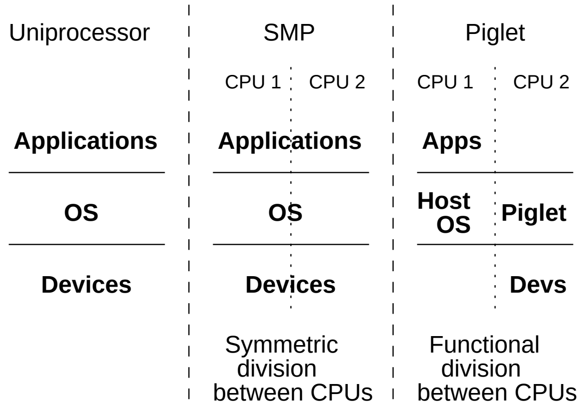

Figure 1: Asymmetry in Piglet

One model which has been proposed and appears to offer solutions to these problems is the vertically structured operating system e.g. the University of Cambridge's Nemesis project [1], and MIT's Exokernel [2].

These operating systems provide only very low-level services in the OS kernel (typically safe multiplexing of hardware between applications) implementing all other functionality e.g. scheduling, in user-space, possibly by shared libraries. This allows applications to easily tailor resource management policies to their own needs.

While the vertical structure promises to solve many of the problems described above, it requires application designers to become familiar with a whole new system model. We believe that many of the benefits of vertical operating systems can be provided to applications without imposing a new system model. We have designed and implemented an asymmetric multiprocessor OS named Piglet to investigate these ideas.

2 The Piglet architecture

Piglet uses multiple processors, possibly of different architectures and capabilities (e.g. per-NIC processors and the central processor(s) of a switch or router), in an asymmetric manner as shown in Figure 1.

The system CPUs are partitioned into *functional* *groups*, each consisting of one or more processors. The host processor group runs the host OS e.g. Linux, and all user applications. Additional functional groups are then dedicated to the management of a certain class of devices, such as network interfaces. The host OS and applications (which have direct user-space access to devices in the Piglet architecture) then access those devices through one or more *Virtual Device Interfaces, VDIs.*

Each processor within a functional group runs the *Lightweight Device Kernel, LDK,* a kernel-mode process which manages the associated devices and implementations of those functions of the VDIs which are not directly available in hardware.

For example, we may provide a VDI which performs TCP/IP checksumming of network data for the user application. If the NIC which data will be sent on provides this functionality in hardware then we can make use of that functionality. If, however, this function is not available, then the LDK can calculate the checksum, thus maintaining a uniform VDI. This means that new devices can be rapidly brought into the Piglet architecture without changing the system interface.

The system can be dynamically switched between different divisions of the CPUs — if computational power is needed then all the CPUs can be placed into the host processor group, giving a traditional SMP structure. Only when it becomes advantageous to do so are CPUs dedicated to a functional group.

3 Resource management using Virtual Clock

While many network-oriented applications do not have specific resource requirements e.g. FTP, e-mail, browsing the World Wide Web, those applications which process continuous media, such as digital audio and video, need to be guaranteed certain resources if they are to provide the end-user with an acceptable rendering of their data.

A bursty application, such as a Web browser, can wreak havoc on the transmission of digital video over the network if the video application is not guaranteed the bandwidth it requires to sustain an acceptable quality display.

Traditional operating systems cannot easily provide the kind of resource guarantee required by video transmission because their multiplexing and demultiplexing of network data is done at too high a level. For example, the TCP/IP stack usually multiplexes all the TCP connections associated with a single interface onto one IP stack.

Vertical operating systems solve this problem by multiplexing[2] network streams onto the network interface at the very lowest level, each application having its own TCP/IP stack.

Piglet provides the same low-level multiplexing as vertical operating systems, each application being allocated its own network queue within the VDI. If applications access these queues directly then the structure is very similar to a vertical OS. Even if applications still access queues through the kernel, thus using only a single IP stack, we can still provide many of the benefits by scheduling the movement of packets to and from this single stack.

Piglet uses Virtual Clock[3] to schedule the delivery of packets from queues to the network interface and vice-versa. It allows an application to specify its required throughput and traffic burstiness in terms of the amount of data to be sent per time period. In addition, priority is given to those applications which have specified resource requirements over those which have not. Queues associated with the latter class of application are only serviced if none of the guaranteed-bandwidth queues have pending data.

Our experiments, described in more detail in [4], show that this mechanism can provide applications with guaranteed network bandwidth[3] in the face of applications which try to use as much bandwidth as possible.

4 Pipelined Protocol Stacks

While the level at which multiplexing is performed in a protocol stack affects the ability of the OS to provide resource guarantees, the protocol stack structure affects its throughput. Traditional operating systems have highly serial implementations of protocol stacks corresponding to their layered architectures, as shown in Figure 2.

While layering provides benefits in terms of data abstraction, this implementation scheme fails to take advantage of the concurrency available in the protocol stack (it can be viewed as a type of dataflow). Each operation is typically independent of every other, and a packet can also be handled without reference to any other. These properties mean that many functions of a protocol stack may operate in an overlapped or pipelined fashion, as shown in Figure 3.

[2] From the purposes of this discussion the same arguments apply to demultiplexing.

[3] On the sending host at least; local mechanisms do not

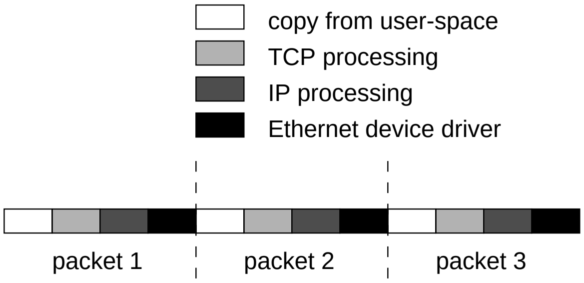

Figure 2: Serial TCP/IP stack implementation

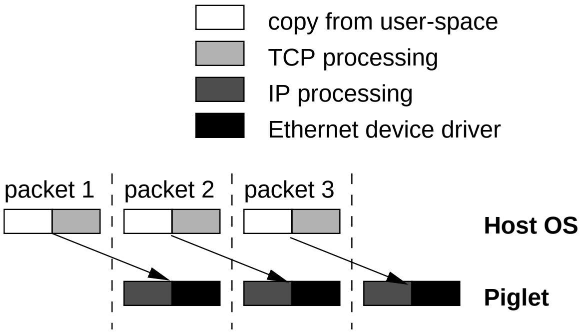

Figure 3: Pipelined TCP/IP stack implementation

Although such a protocol stack could be implemented in a traditional SMP operating system, working within the context of existing protection boundaries and thread/process scheduling would be challenges; the functionally-divided structure of Piglet allows us to more easily move part of the protocol stack into the LDK. The Virtual Device Interface uses shared-memory queues to move data between the application (or host OS) and Piglet, thus providing a clean abstraction and concurrency control between the two 'functional units' (by analogy with a pipelined CPU).

It is also relatively straightforward to split the protocol stack at different boundaries and provide multiple VDIs to the LDK, allowing the application to select the most appropriate functional split. However, splitting the protocol stack in the manner shown in Figure 3 is attractive in that IP processing and lower levels need never examine the packet payload, thus eliminating the need to move that payload between the two processors' D-caches.

5 Piglet Services

We can further extend this model of pipelined protocol stacks by adding application-specific *services* to the protocol stack. If we regard the application's data-processing functions as extra layers in the protocol stack then the Piglet pipelined structure can be extended into the application; Piglet *services* provide various functions to applications in this manner.

help deal with 'unruly' hosts.

5.1 Service Structure

Each network flow within Piglet may be associated with a service which performs processing of packets sent and received on that flow. A Piglet service is implemented in three parts: a transmit function, a receive function, and one or more threads:

- The transmit function takes packets from the flow's queue and performs some service-specific processing on those packets before passing them to the appropriate network interface.
- The receive function performs the reverse processing, except that because it runs in the receive interrupt handler it must be fast. Hence, the receive function passes the packet to the service's threads which perform the necessary processing.

Since services are currently statically compiled into the Piglet LDK, we do not believe that it is necessary to enforce protection barriers between the threads and LDK. Threads execute in the same address-space and at the same privilege level as the LDK, and are non-preemptible (although they can be interrupted). This makes context switching between threads very fast (only 10 instructions on the current x86 implementation) and simplifies concurrency control.

5.2 The Piglet MPEG Decoder

As an example of a non-trivial service which extends the pipelined protocol stack structure to an application we have implemented an MPEG-1 decoder as a Piglet service. The decoder is based upon the Berkeley *mpeg_play* application [5] which displays MPEG-1 streams in an X window. We modified mpeg_play to transmit MPEG frames over the network using a simple UDP-based protocol.

The MPEG-1 decoding process (as implemented by mpeg_play) can be conveniently split into two phases: decoding of MPEG-1 frames into a (Y, U, V) representation of an image, and conversion from (Y, U, V) format to (R, G, B) for display (this phase may also include dithering of the image to reduce color depth).

Decoder	Thorax (fps)	Links (fps)
Application (mpeg_play)	17.5	22.5
Piglet	19.2	25.0

Table 1: Reported frame-rates for MPEG decoding

Since both phases are computationally intensive, there is obvious benefit to be obtained by performing them in parallel. Additionally, the structure of the two phases makes them ideally suited to pipelining. Hence, the Piglet MPEG-decoding service performs the decoding of MPEG-1 frames into the (Y, U, V) representation, leaving the application performs conversion to (R, G, B) and dithering.

In addition to increasing concurrency, this permits the substitution of different video coding protocols without requiring that the application be changed, and also allows Piglet to control the rate at which frames are made available to the application (either for synchronisation/clocking purposes or for resource management).

If each image (typically about 112kB for 320×240 image) was passed to the application as a sequence of data packets then the overhead of processing these packets, including copying of data, would far outweigh the benefits obtained from pipelining. Instead, Piglet's memory buffers into which images are decoded are directly accessible by the application, eliminating most of these overheads.

5.3 Decoder Performance

To illustrate the feasibility of this approach, the Piglet decoder was tested on two video streams: *thorax* [6] and *links* [7]. The former consists of 1356 frames totalling 5.05MB, while the latter is 80 frames/155kB. The average frame-rate reported by mpeg_play is shown in Table 1.

The first row of the table gives the frame rates for the standard mpeg_play application modified to transmit the MPEG stream over the network. Piglet plays no part in this situation, the application performing decoding, conversion, dithering and rendering.

The second row shows the frame rate when Piglet performs the MPEG decoding and the application only performs conversion, dithering and rendering. We see that the frame-rate for both streams is about 10% higher. We believe that with an MPEG player designed specifically as a front-end to a Piglet decoder, rather than modified from a standard player, this figure could be further increased.

6 Conclusions

We believe that Piglet presents an attractive architecture for an OS designed to support networked continuous media for the following reasons:

1. Resource guarantees can be made to applications without changing the host OS. Virtual Clock allows us to guarantee network bandwidth to applications, regardless of other applications.
2. A greater degree of concurrency is achievable more simply than in a traditional OS, thus making more efficient use of multiple processors.

Our implementation of an MPEG decoder as a Piglet service demonstrates that this greater degree of concurrency can provide quantitative benefits to applications processing continuous media.

References

[1] 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.

[2] Dawson R. Engler et al., *"Exokernel: An Operating System Architecture for Application-Level Resource Management"*, Proceedings of the 15th ACM Symposium on Operating Systems Principles (December 1995).

[3] 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.

[4] Steve Muir and Jonathan Smith; *"Functional divisions in the Piglet multiprocessor operating system"*, to be published at SIGOPS European Workshop, Sintra, Portugal (September 1998).

[5] Lawrence A. Rowe et al., *mpeg_play (MPEG Video Software Decoder)*; Berkeley Multimedia Research Center, 1995 **http://bmrc.berkeley.edu/projects/mpeg/**

[6] Kiernan Holland, *"Rise of the Thorax"*; 1995 **http://www.unm.edu/~kholland/ /khsplitC3.html**.

[7] Author unknown,*links* animation, **http://www.mpeg1.de/movies/links.m1v**