Better mousetraps

John Wilkes

Hewlett-Packard Laboratories, Palo Alto, CA

Position paper for the 1994 ACM SIGOPS European Workshop on "Matching operating systems to application needs"

As a technologist, working in the field of storage systems, and employed by a company that develops and sells a wide range of computer systems and components, I've learned that the biggest impediments to technology transfer—including better OS support for specific applications—are not technical, but economic. This paper summarizes some of these issues, and considers how best we might respond to them.

1 How did we get where we are today?

Applications can be divided into three rough classes:

1. *technology-limited*: where we simply do not have solutions that work well at acceptable cost (e.g., "grand challenges", portable digital assistants (PDAs) with problematic battery-weight-to-power ratios).
2. *price-limited*: solutions exist, but their cost is such that better price-performance is a competitive advantage. The application that determines the size of a system that a customer needs to buy is usually in this category.
3. *well-served*: most applications have needs that are modest enough to be met by the standard services.

1.1 OS development is an economic endeavour

Developing operating systems, like developing applications, is an expensive business. A better OS is just one of many opportunities in which a company can invest.

Since most commercial OS vendors are driven by business needs, not technology availability, there is pressure to maximize the size of the well-served application set. This often results in trade-offs that would appear non-optimal from a purely technical standpoint. For example, leaving the database writer's job a little harder for a little longer to let a new application be supported has often been a good business trade-off.

Occasional forays are also made into satisfying the needs of the price-limited applications for sufficiently lucrative sales opportunities. (For example, fast locking primitives to support a popular third-party database.)

Of course, this three-way problem division is subject to all sorts of assumptions and caveats. For example, not all users put the same value on their applications, or have the same amount of money to spend. Consequently, an application that may be considered well-served in one environment may fall into the price-limited class for another. This also makes the job of providing a single OS across a wide hardware range much harder: engineering for high-end needs costs the users at the low end more—both in terms of development costs and (often) runtime overheads such as memory requirements.

Until there is economic benefit from supplying a feature, there is no incentive for providing it. Only once this gap has been crossed do technical issues such as the quality and cost of the solution matter.

1.2 Concurrent changes are hard

Both the OS and the applications have to change before any benefit is obtained from changing the abstractions at the OS–application boundary. This makes it hard for commercial enterprises that do not control the code on both sides of the interface to make what seem like obvious, desirable improvements.

For example, OS-specific optimizations are hard to justify for a database vendor that is trying hard to keep their product portable across a wide range of platforms. At the same time, the OS vendors are also trying to keep multiple database vendors happy: again, each investment has only a limited

return because it is only applicable to a subset of the market for the OS.

1.3 Despite technology advances, price/performance still matters

In a world where people pay real money for solutions to their problems, there are commercial advantages to being able to provide the same solution for less money (all other things being equal, of course!). This translates into a need to continue to pursue price/performance improvements, despite enormous progress on the underlying hardware.

Some applications are so driven by specialized requirements that dedicated OS support is accepted as a necessary cost. For example, in a video-on-demand server, the disk costs dominate the server cost, so maximizing disk utilization is an overriding concern.

Because of this, divide-and-conquer specialization is going to grow in importance: the "one size fits all" approach cannot effectively be applied to an ever-widening range of problems. Fortunately, dedicated servers in a network are an ideal delivery vehicle for this specialization.

1.4 Abstractions limit control

Abstractions (interfaces) have to balance expressive power (control) and ease of use (simplicity). A good solution for the well-served class of applications may be a poor one for particular price-performance limited ones.

Consider the switch between IBM's OS/360-style file-access primitives and those of the UNIX system. The first of these is an abstraction of the physical storage devices that used to be directly visible to applications. But once systems got sufficiently faster that many applications did not need such control, coupled with the technical innovation of a buffering file system, we were able to move to the byte-stream abstraction. Databases have been coping with the resulting loss of control ever since.

[Keppel93] has a nice analysis of this issue.

2 How should we respond?

As a group of computer scientists, we can help with technology development; deployment requires more, and is what distinguishes our field from a pure science. Here are a few aphorisms we can live by during this process, driven by the economics-based observations above.

- *OS research should better support specialized systems*

No one OS vendor (or even OS) can afford to support all the functions needed from tomorrow's applications. Trying to do so using current structuring techniques will only add to the bloat.

In these circumstances, an OS that allows a vendor or customer to *configure* a system to meet a particular need is the way to go. The more easily configured, and the more effective the configuration changes, the more advantage will such systems have. To be able cover a wide range of application needs, some defined framework has to be developed and built on (e.g., [Campbell92, Wilkes93]).

- *The purpose of operating systems is resource multiplexing, not code sharing*

Successful specialization across a wide range of needs requires a much more minimalist approach to the base framework than has been taken by most "microkernel" efforts to date. We need to revisit this issue now that context-switches can be done in a few microseconds or less.

- *Use declarative specifications*

The number and range of specifications that will have to be supported continues to grow. The best (maybe only) way to handle this is to give applications a way to express their requirements in a declarative form (e.g., [Gelb89, Wilkes91]).

This has several benefits: it leaves much greater freedom for the underlying services than does an imperative implementation of a policy; it separates the "extras" from the basic functional interface, allowing each to be simpler ([Keppel93] refers to this as the meta-control approach); and it concentrates the application writer on precise specifications of their needs.

- *Augment declarative specifications with adaptive techniques*

Having an (OS) implementation adapt its behavior as a function of the observed access patterns to it is a powerful technique. It relies on the past being a good predictor of the future, but this seems to be true surprisingly often.

- *Requirements are not optional*

Finally, we *do* have to develop and refine some new abstractions. Adaptability is a powerful tool for adding performance while keeping an interface unchanged, but nothing involving guarantees can be deduced after the fact—in particular availability,

predictability (realtime), and bandwidth needs. These must all be specified in advance.

With suitable information, today's machines are quite capable of meeting these needs, but our abstractions are failing us: having occasional high-bandwidth disk access is not the same as a guaranteed-rate display of a video stream.

What we are witnessing today is the shaking out of the essential requirements from the "optional" ones. It is being complicated by the concurrent emergence of new technology such as Gbit/s networks, which will go a long way to simplifying the implementations of such facilities. (In fact, the new "enabling" technologies are causing some confusion, as people sometimes forget the differences between guarantees and mere speed.)

3 Conclusion

My own field is storage subsystem design, with particular interest in a number of specialized applications (OLTP, video-on-demand), as well as for general-purpose computing. The approaches described in this position paper are all proving useful in reducing the costs of providing good solutions for both kinds of application—and hence increasing the utility of those solutions to customers.

The impact of a workshop like this one could well spread beyond the immediate participants if it can help propagate such approaches, in addition to identifying new technologies.

Acknowledgment

Richard Golding provided much helpful feedback on this paper, and introduced me to the work of David Keppel.

References

[Campbell92] Roy H. Campbell, Nayeen Islam, and Peter Madany. Choices, frameworks and refinement. *Computing Systems* **5**(3):217–258, Summer 1992.

[Gelb89] J. P. Gelb. System managed storage. *IBM Systems Journal* **28**(1):77–103, 1989.

[Keppel93] David Keppel, Susan J. Eggers, and Robert R. Henry. *Evaluating runtime-compiled value-specific optimizations.* Technical report 93–11–02. Department of Computer Science and Engineering, University of Washington, November 1993.

[Wilkes91] John Wilkes and Raymie Stata. Specifying data availability in multi-device file systems. *Position paper for 4th ACM-SIGOPS European Workshop* (Bologna, 3–5 September 1990). Published as *Operating Systems Review* **25**(1):56–59, January 1991.

[Wilkes93] John Wilkes. DataMesh, house-building, and distributed systems technology. *Proceedings of 5th ACM SIGOPS European Workshop* (Mont Saint-Michel, France, 21–23 September 1992). Published as *Operating Systems Review* **27**(2):104–108, April 1993.