Open Source as a Foundation for Systems Research

Muli Ben-Yehuda muli@il.ibm.com
Eric Van Hensbergen ericvanhensbergen@us.ibm.com

"Pigmaei gigantum humeris impositi plusquam ipsi gigantes vident"[1]
—Sir Isaac Newton

Introduction

You are a systems researcher at a corporate research lab. The corporation you work for deals with both proprietary and open source software. You have an exciting new idea that will undoubtedly revolutionize the field, but first you need to build a working system to validate it. Before embarking on your exploratory research project, you must decide: Do you start from scratch, or do you build upon a mature system? And if the latter—should the system be proprietary or open source?

The first question, "do you start from scratch?", is pretty easy to answer. Assuming your idea can be expressed as an evolutionary (or even revolutionary) change to an existing system, far reaching though it may be, building on an existing system helps jump-start the process. Starting from first principles is often appealing from a puristic point of view, but the reality of schedules and external pressures is such that it is good to be able to get up to speed and show preliminary results quickly. The rest of this paper deals with the second question, namely "should the foundation for your work be a proprietary or an open source system?".

We posit that mature open source systems are a perfect match for exploratory research efforts. The gist of our argument is that open source culture and processes are a natural fit for exploratory research efforts, whereas proprietary code culture and processes often present more barriers than benefits to exploratory efforts. Open source provides an excellent platform upon which to build the exploratory research as well as a venue for wide-spread deployment of experimental mechanisms to a large audience. The infrastructure is often critical to jump-starting an exploratory project, in order to be able to focus on the important bits. It is particularly suitable for exploratory research efforts since the end-users are free to reconfigure and rebuild the entire system, allowing them the freedom to configure (in or out) even invasive exploratory components. To do the same in proprietary systems would require a more complicated runtime configuration infrastructure or potentially even alternate versions of binaries.

[1]Commonly translated as "if I have seen farther, it is by standing on the shoulders of giants."

Proprietary-code projects almost always have tight code release schedules and the demands of a-priori review and testing of all code that is incorporated often prohibit the integration of more exploratory components. By contrast, open source distributes codes review and verification responsibilities across the entire community, promoting a natural balance between new contributors and review workload. Another barrier within proprietary projects is that they often employ proprietary tools, build, and even execution environments. This increases both the financial cost and setup overhead for the researcher. Successful open source projects rely on simple, reproducible build environments in order to promote collaboration. If a build process is too cumbersome or the execution environment too obscure, the size of the open source community for that project is limited, often leading to stagnation or migration to competing projects with lower barriers to participation.

In summary, open source projects tend to evolve toward a culture promoting inclusion. While proprietary-code projects can be structured to promote exploratory work, it has been our personal experience that the structure and bureaucracy which proprietary-code culture instills often stands in the way of synergistic activity.

So far we have shown that open source projects provide the right foundation for exploratory efforts. The curious reader might wonder, however, whether open source projects stand to gain anything from being the foundation for exploratory efforts, and whether they should actively encourage it? We believe the answer is a resounding "yes!". To see why, consider that the exploratory effort offers the open source project a possible path forward, with a more or less clear notion of what works and what doesn't. If the exploratory effort proves successful, it could even offer a design and an implementation that may be directly incorporated into the open source system. Thus, through incorporation of lessons, ideas, design and code from the research effort, the open source project can make progress unhindered by its legacy.

IBM's corporate involvement in open source technology and collaboration has nurtured IBM research's ability to use open source as a platform for exploration. In support of our argument that open source is the right foundation for exploratory research efforts, we present the following two projects which have been recently carried out by the authors, the v9fs project and the Calgary IOMMU project.

The v9fs Project

IBM's interest in the v9fs project [10] was originally motivated as a testbed for evaluating 9p [8, 16] as a unified lightweight mechanism for sharing resources and services between logical partitions in a virtualized environment. The client code was based on a previously abandoned effort by Los Alamos National Labs to add a 9p client to the 2.4 Linux kernel. In addition to the foundations for the client code, Linux versions of the 9p server applications were provided entirely from other open source projects from Vita Nuova [17] and MIT [5]. Use of existing open source projects as a base allowed for accelerated prototyping and evaluation of the concepts without extensive development effort.

After the Linux 2.6 client prototype was complete, the decision was made to try to get it merged into the mainline kernel. The process involved almost a complete line-for-line rewrite of the code to try and meet community coding style guidelines. The initial re-write was done as a spare time project over 5 months, and then another 6 months was spent between the time the initial patches were sent out (against version 2.6.11) and when the code was finally included (in kernel version 2.6.14). The Linux kernel community code review process discovered (and continues to discover) dozens of corner case error conditions and implementation bugs resulting in a more mature implementation than would have typically been present in a research prototype.

The presence of the 9p client within the mainline kernel renewed interest within the national labs, and the original authors of the 2.4 client picked up the 2.6.14 client and began integrating it into their cluster infrastructure [15]. Additionally, a community member was inspired to author a new Linux 9p server infrastructure which allowed easy construction of user-space file servers. This same infrastructure was later adopted by IBM as part of its Libra project [1, 9] allowing us to share more complicated resources and services between logical partitions. More recently, other enthusiasts have begun work on adding enhanced cache support, authentication, and other facilities. Additionally the v9fs code is being used as part of several projects and application infrastructures [14]. The native Linux 9p client also inspired a BSD client, a FUSE client, as well as a dozen language bindings including ones for Ruby, C, Python, Java, Caml, LISP, and TCL. IBM's involvement in the v9fs project created an ongoing relationship with LANL and Sandia National Labs leading to collaboration on several other mutually beneficial research activities [6, 11].

The Calgary IOMMU project

An I/O Memory Management Unit ("IOMMU") creates one or more unique address spaces, which are then used to control how a DMA operation initiated by a device accesses memory. IOMMUs have long been used to address the disparity between the addressing capability of some devices and the addressing capability of the host processor. As the addressing capability of those devices was smaller than the addressing capability of the host processor, the devices could not access all of physical memory, and the IOMMU was used to map the limited device address space into the larger physical memory address space (e.g., from 32-bit to 64-bit).

IOMMUs with multiple distinct address spaces, commonly referred to as "isolation-capable IOMMUs", can restrict a device such that the device can only access pre-assigned areas of physical memory. Without isolation, a device controlled by an untrusted entity (e.g., a virtual machine running on top of a hypervisor or a non-root user-level driver when running on bare-metal) could compromise the security or availability of the system by corrupting the machine's memory [3, 7, 13].

IBM, by virtue of having a common chipset in some of its PowerPC and x86 based servers, was the first x86 server vendor to build an x86 server with an isolation-capable IOMMU. The Calgary IOMMU project was initiated in order to understand the challenges inherent in adding operating system and hypervisor support for an isolation-capable IOMMU on the x86 platform. Considering the project goals, writing our own OS or hypervisor were too much of a stretch. Therefore, for all of the reasons presented in the introduction, we picked an open source OS (Linux) and hypervisor (Xen [2]) as a basis for our work.

The first stage of the project was to verify that the hardware worked and learn about the interaction between the operating system, the hypervisor and an isolation-capable IOMMU. This inherently exploratory stage was significantly helped by basing it on a mature OS, Linux, which on x86 included some—albeit not all—of the necessary facilities [3] for isolation-capable IOMMU support.

Once we prototyped IOMMU support and everything more or less worked, we made a crucial decision: To submit the patches upstream to be included in the Linux kernel. Even though we had to take some time to get the patches up to the Linux kernel style and quality, having the prototype included in the Linux kernel had several benefits: First, it resulted in an immediate boost in quality (due to the "many eyeballs" effect [18]). Second, it gave interested customers immediate access to a useful feature (isolating mis-behaving adapters), and third, it exposed numerous shortcoming in the Linux kernel which were subsequently fixed by both the authors and the community. Last but not least, it also gave us a concrete implementation to base future design decisions on.

Next we moved on to adding isolation-capable IOMMU support to the Xen hypervisor. During the course of the Xen implementation, numerous bugs were discovered and fixed in Linux kernel Calgary IOMMU implementation. Those bugs which were also applicable to the Xen environment (e.g., bugs related to the hardware interface) were immediately propagated to the Xen code-base. Thus the exploratory effort benefited from the mature work without being hindered by it.

How did the mature system benefit from the exploratory effort in this case? We note that Linux has recently gained the ability to act as a hypervisor, via both the *kernel-based virtual machine (KVM)* project [12] and the *lguest* project [19]. Both will require in the near future a para-virtualized IOMMU interface. The exploratory effort done in the context of the Xen Calgary IOMMU implementation provides valuable lessons to guide the design and implementation of such an interface in Linux [4].

Conclusions

We have shown two separate research efforts, both of which greatly benefited from being based on mature open source projects, to the extent that neither effort is likely to have have been successful had it been started from scratch or based on a proprietary code base.

This leads us to believe that systems research is best conducted in an open source environment where exploratory and mature components can easily coexist, leading to broader review and use of the research. This wider audience often leads to new challenges and ideas, catalyzing future research. As such, institutions should consider rewarding open source contribution in a similar manner to publication as they are both peer-reviewed, relate to the broader dissemination of knowledge and advance the state of the art.

REFERENCES

[1] G. Ammons, J. Appavoo, M. Butrico, D. D. Silva, D. Grove, K. Kawachiya, O. Krieger, B. Rosenburg, E. V. Hensbergen, and R. W. Wisniewski. Libra: a library operating system for a jvm in a virtualized execution environment. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 44–54, New York, NY, USA, 2007. ACM Press.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[3] M. Ben-Yehuda, J. Mason, O. Krieger, J. Xenidis, L. Van Doorn, A. Mallick, J. Nakajima, and E. Wahlig. Utilizing IOMMUs for virtualization in Linux and Xen. In *OLS 2006: Proceedings of the 2006 Ottawa Linux Symposium*.

[4] M. Ben-Yehuda, J. Xenidis, M. Mostrows, K. Rister, A. Bruemmer, and L. Van Doorn. The price of safety: Evaluating IOMMU performance. In *OLS 2007: Proceedings of the 2007 Ottawa Linux Symposium*.

[5] R. Cox. Plan 9 from user space. `http://swtch.com/plan9port`.

[6] C. Forsyth, J. Mckie, R. Minnich, and E. V. Hensbergen. Petascale Plan 9. USENIX 2007 Poster.

[7] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Reconstructing I/O. Technical report, University of Cambridge, Computer Laboratory, August 2004.

[8] E. V. Hensbergen. Plan 9 remote resource protocol RFC, 2005. `http://v9fs.sourceforge.net/rfc`.

[9] E. V. Hensbergen. Partitioned reliable operating system environment. *Operating Systems Review*, 40(2), April 2006.

[10] E. V. Hensbergen and R. Minnich. Grave robbers from outer space: Using 9P2000 under Linux. In *Proceedings of the 2005 UNIX Annual Technical Conference*, 2005.

[11] L. Ionkov, A. Nyrhinen, and A. Mirtchovski. CellFS: Taking "DMA" out of cell programming. `http://www.xcpu.org/cellfs-talk.pdf`.

[12] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux virtual machine monitor. In *OLS 2007: Proceedings of the 2007 Ottawa Linux Symposium*.

[13] J. LeVasseur, V. Uhlig, J. Stoess, and S. Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec. 2004.

[14] K. Maglione. Window manager improved 2. `http://www.suckless.org/wiki/wmii`.

[15] A. Mirtchovski and R. Minnich. XCPU: a new, 9p-based, process management system for clusters and grids. In *Proceedings of the 2006 IEEE International Conference on Cluster Computing*, 2006.

[16] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, Summer 1995.

[17] R. Pike, D. Presotto, S. Dorward, D. M. Ritchie, H. Trickey, and P. Winterbottom. The Inferno operating system. *Bell Labs Technical Journal*, 2(1), Winter 1997.

[18] E. S. Raymond. *Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001. Foreword by Bob Young.

[19] R. Russel. lguest: Implementing the little Linux hypervisor. In *OLS 2007: Proceedings of the 2007 Ottawa Linux Symposium*.