Beastie Meets Raccoon: MINIX 3 as a BSD

[Extended Abstract]

Ben Gras
Dept. of Computer Science,
VU University Amsterdam,
The Netherlands
ben@minix3.org

Gianluca Guida
Dept. of Computer Science,
VU University Amsterdam,
The Netherlands
gianluca@minix3.org

Arun Thomas
Dept. of Computer Science,
VU University Amsterdam,
The Netherlands
arun@minix3.org

Thomas Veerman
Dept. of Computer Science,
VU University Amsterdam,
The Netherlands
thomas@minix3.org

ABSTRACT

MINIX 3 has imported a significant amount of userland BSD code. The trend began several years ago, but the pace has quickened markedly. We have already imported NetBSD's buildsystem, NetBSD's C library, the pkgsrc package management infrastructure, and various userland utilities from NetBSD and FreeBSD. We are currently in the process of porting a full NetBSD userland as well as puffs for increased filesystem support. Though not technically BSD code, we have adopted clang/LLVM as our default toolchain, and we are working to adopt elftoolchain as a binutils replacement.

When MINIX 3 was originally conceived, the goal was to create a robust multiserver operating system that maintains POSIX compatibility. We leveraged our multiserver architecture in which most OS code runs in separate usermode processes to provide new functionality, such as driver isolation and restartability. Now, we would like to push farther than just POSIX compatibility and provide a system that looks much like a BSD from a user's perspective. This paper serves as a progress report on our ongoing work turning MINIX 3 into a BSD. We have a long way to go before MINIX implements all BSD functionality, but we have a good start. We will continue to pull in BSD code, and we have identified future opportunities to pull in driver code and kernel components from BSD.

This will serve to enable our vision of the best of both worlds: isolation and restartability features unique to Minix combined with the well-maintained, real-world-hardened system code for drivers, filesystems, userland, and other OS code, of a modern BSD OS.

1. INTRODUCTION

MINIX 3 is not a BSD officially, but it is starting to look a bit like one. MINIX has been steadily importing userland code from NetBSD. The ultimate goal is to make MINIX as BSD-like as possible from the user's perspective, while maintaining our multiserver architecture and our unique fault tolerance features. To this end, we have adopted the pkgsrc package management infrastructure as well as the NetBSD C library and userland. We are also working on importing the puffs userspace filesystem infrastructure from NetBSD.

This paper will describe the design tradeoffs and implementation issues we encountered while incorporating these BSD components into our system. We hope our experiences will be useful to other OS projects who are incorporating BSD code. We also hope the BSD community will find it interesting to hear how their code is being used and adapted.

2. BACKGROUND

2.1 A Brief Introduction to MINIX 3

This section provides a brief introduction to MINIX 3 and its multiserver architecture, in which most OS code runs in separate usermode processes. We will describe past work in the area of driver isolation fault and driver restartability [4, 3, 10, 5, 6]. We will talk about the design choice to maintain POSIX compatibility. We will provide enough background so the reader can follow the rest of the paper.

2.2 MINIX 3's March Toward BSD

This section will provide a brief history of how BSD code made its way into MINIX 3. The process started with the occasional userland utility, library functions, and header file being imported into MINIX. Then, when MINIX grew to need a more advanced buildsystem, we adopted the NetBSD buildsystem. This made it even easier to import more userland utilities. We then adopted pkgsrc. In order to support pkgsrc, we imported yet more BSD userland code and library functionality. Then, we worked to import the NetBSD C library. About the same time, we worked to migrate to the clang/LLVM toolchain. Currently, we are importing puffs and the entire NetBSD userland.

This section will also briefly discuss some of the philosophical similarities between MINIX 3 and BSD. MINIX 3 is licensed under the BSD license. Like the BSDs MINIX 3 ships a full OS distribution, not just a kernel. Several MINIX 3 developers are longtime BSD people.

DRAFT: as of May 30, 2011

3. PKGSRC

Minix has successfully adopted pkgsrc as the primary package management system. Most of the work for this was performed by Gautam Tirumala (mentored by Arun Thomas and Ben Gras) during his GSOC 2010 tenure and integrated immediately.

We only build a small fraction of the packages in pkgsrc currently, but find insofar as there are no fundamental features missing from the OS, we can build with minimal changes. The two major missing features blocking many packages are missing pthreads and shared library support.

Minix provides a pre-bootstrapped pkgsrc environment similar to what NetBSD does. Packages fundamental to pkgsrc (i.e. pkg_install) come pre-installed with Minix installations, and people can install pre-built binary packages out of the box with pkgin. They can easily get the pkgsrc source hierarchy using git (we use DragonFly's git mirror of the pkgsrc cvs tree) if they wish.

This year's GSOC, performed by Thomas Cort and mentored by Gautam and Ben, will be another big step for pkgsrc on Minix, most significantly bulk building and upstreaming Minix support, hopefully leading to official Minix support in pkgsrc.

4. NETBSD C LIBRARY AND USERLAND

The Minix project is in the process of adopting the NetBSD libc, entirely replacing its own libc, as part of the strategy of modernizing Minix system software that is not Minix-specific. Implicitly this means: also adopting the NetBSD C headers hierarchy, and replacing the one Minix has. The goal is to have a modern, well-maintained libc plus C headers set, to lower the barrier to running modern application programs.

A significant secondary benefit to this work is that we have a succinct overview of shortcomings of the Minix system call API - i.e. missing features and system calls that are required to support a modern libc. This is due to the fact that we decided to maintain a separate in-source-tree 'patch' file, spelling out the differences between the original NetBSD code and its port to Minix. It has become an ongoing goal of the Minix project to reduce this patch file, and other similar patch files, and so reduce the maintenance burden of our libc.

There were some hurdles in porting the NetBSD libc. One surprising finding was that support for the case of 'aout and non-reentrant' had deteriorated to a degree, causing compile problems. Another is the fact that the kernel part of the NetBSD C headers are from a different hierarchy than the standard includes, an organisation significantly different from the Minix headers. Furthermore, there is quite some minix-specific code that had to be kept, both in libc and headers code. Also the Minix networking API is sufficiently different from the standard BSD sockets interface that much of that code had to be different.

Currently we are migrating the base system and pkgsrc packages from the Minix libc/headers system to the NetBSD one. Much base system code contains cases for both to smooth the migration path.

The end result of the netbsd libc project is that a relatively clean port was done, with which we can compile many BSD userland utilities unmodified.

5. PUFFS

The goal of porting puffs [7] is to ultimately be able to port librefuse [1] and be able to run Fuse file systems. In MINIX 3, the entire file system already runs in user space, which should make the process easier.

We have a Virtual File System (VFS) running on top of several File Servers (FS). By default MINIX 3 is installed using three sub partitions with the MINIX 3 file system; each (sub) partition is handled by an FS instance. VFS communicates with these FSes by sending messages using the VFS-FS protocol, which is essentially a set of Posix-like calls (e.g., mknod, chown, link, unlink, etc). One of the challenges will be to translate this mechanism to the PUFFS equivalent.

For example, puffs operations work on file paths, where VFS-FS requests work on inodes. Also, file systems implemented using PUFFS receive mount parameters through argv[], but in MINIX 3 there is a separate VFS-FS request that provides this information. Moreover, in MINIX 3 every driver and system server has to deal with the System Event Framework (SEF) which handles live updates and failure recovery. In particular, during the start of a driver some SEF parameters have to be set before it starts its main loop.

While work is underway to "decouple" VFS from FS instances by making communication asynchronous, there is currently no support for subsequent requests to the file system by a puffs instance. For example, at the moment it is impossible to port sshfs to MINIX 3, as talking to an sshfs instance will ultimately result in having to talk to INET to get access to the network. This communication has to pass through VFS which will result in a deadlock, because VFS is still stuck talking to the sshfs instance. We do have some support for callbacks to VFS, but that is too limited to handle this problem. However, it will be solved with the introduction of the asynchronous VFS, which only has to prevent PUFFS instances (or FSes, for that matter) from talking to itself.

6. TOOLCHAIN WORK

Since its first release, MINIX has used the venerable Amsterdam Compiler Kit (ACK) [11] as its toolchain. ACK provides a C compiler, assembler, linker, archiver, and other binary utilities. It is roughly equivalent in functionality to the combination of GCC [12] and binutils [13]. Since ACK does not support *long long* and other C99 functionality, we have decided to adopt Clang/LLVM [9] as our base system compiler. In the past, we would apply ACK-specific workarounds when importing BSD code. This is not the best solution. Now that we have a more modern toolchain, we no longer have to apply these workarounds.

While we have support for compiling our base system with GCC, we prefer BSD-licensed components so we would rather use Clang/LLVM as our default compiler. We are also interested in Clang/LLVM because of its fast compile times. Additionally, we are interested in adopting KLEE [2], a symbolic execution tool built on LLVM, to help us verify our software. We will continue to use GCC to build our pkgsrc packages for the time being, though we are keenly following the progress of Clang/LLVM pkgsrc work.

We are currently using clang/LLVM r125950 from pkgsrc. We have decided not to import clang/LLVM into src, since it would greatly increase the size of our src repository, and we find it simpler to use pkgsrc to manage the toolchain.

In this section, we will discuss some of the problems we encoun-

tered when importing clang/LLVM. Many problems we encountered initially were related to issues with clang's aout support. When clang was originally imported, MINIX did not support the ELF executable format. Now that MINIX supports ELF, many of our aout-specific patches are no longer necessary. There was also a problem we encountered with the clang package requiring libgcc; we resolved this by using compiler-rt from the LLVM project. We have also found that clang is slower than gcc on MINIX; this appears to be a MINIX-specific problem, and we are working to address this.

Our Clang/LLVM port currently requires binutils to work. We have imported libelf from elftoolchain, and we plan to import more of the elftoolchain equivalents for GNU binutils. We, like FreeBSD, are still in need of a BSD-licensed linker. Once we have that, we will have a fully BSD-licensed toolchain.

7. RELATED WORK

This section will provide a brief overview of related work. Apple's Darwin/XNU combines the Mach microkernel with BSD kernel components. We plan on doing something a bit different. We will continue to use our multiserver architecture in which OS components run in separate usermode processes, but we want to incorporate a full BSD userland (C library, utilities, etc). Eventually, we would also like to incorporate more kernel code from the BSDs, but we would like to run these components in usermode via Rump [8].

The Mach microkernel ran BSD 4.3 as its POE server.

LynxOS provides full ABI compatibility with Linux. We are more interested in API compatibility with BSD. We ultimately would like to be able to compile and run all application programs that are supported by NetBSD.

L4/Linux can boot Linux. MINIX 3 is not operating as a hypervisor.

Rump [8] allows you to run BSD kernel code on other operating systems. We would like to leverage it run BSD kernel code, specifically the BSD TCP/IP and driver code, on MINIX 3 in the future.

FreeBSD has Linux emulation support.

8. CONCLUSION AND FUTURE WORK

MINIX 3 is becoming more BSD-like as it imports more code from BSD. We ultimately would like MINIX 3 to look like a BSD from the user's perspective while we continue to do research into the OS core. There is much work to be done before MINIX 3 can reach that state, but this paper describes the work done so far. With the new Libc and userland utilities, this process will become much easier.

We will need to continue to implement missing functionality from MINIX, including missing system calls, shared library support, pthread support. The NetBSD scheduler activations / lightweight process system underpinning its pthreads implementation is being re-implemented for Minix right now, which we expect will enable many missing applications for Minix.

In the future, we are also interested in running NetBSD kernel components on MINIX via Rump [8]. We are especially interested in the BSD TCP/IP stack and driver code. We would also like to contribute more to the upstream BSD community. We have submitted a few patches upstream, but we would like to do much more.

9. ACKNOWLEDGEMENTS

This work has been funded by the European Research Council and the Google Summer of Code Program. We would like to thank our Google Summer of Code students: Gautam Tirumala, Thomas Cort, Evgeniy Ivanov, and Vivek Prakash. We would like to thank the larger MINIX community for their contributions. Finally, we would like to express our utmost gratitude to the BSD community for releasing their well-crafted codebase code under an open-source license, so that we could build on your work.

10. AVAILABILITY

MINIX 3 is available under the BSD license. It can be downloaded from `http://www.minix3.org`. Much of the work described in this paper will be released as MINIX 3.2.0. Progress can be followed in our git tree: `git://git.minix3.org/minix.git`.

11. REFERENCES

[1] A. K. and. ReFUSE: Userspace FUSE Reimplementation Using puffs. In *Proceedings of EuroBSDCon 2007*, pages 29–42, September 2007.

[2] C. Cadar, D. Dunbar, and D. Engler. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 209–224, Berkeley, CA, USA, 2008. USENIX Association.

[3] J. Herder, H. Bos, B. Gras, P. Homburg, and A. Tanenbaum. Reorganizing unix for reliability. In *Proceedings of Asia-Pacific Computer Systems Architecture Conference*, pages 81–94, 2006.

[4] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum. Minix 3: a highly reliable, self-repairing operating system. *SIGOPS Oper. Syst. Rev.*, 40:80–89, July 2006.

[5] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum. Failure resilience for device drivers. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '07, pages 41–50, Washington, DC, USA, 2007. IEEE Computer Society.

[6] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum. Fault isolation for device drivers. In *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '09, pages 33–42, Washington, DC, USA, 2009. IEEE Computer Society.

[7] A. Kantee. puffs - Pass-to-Userspace Framework File System. In *Proceedings of AsiaBSDCon 2007*, pages 29–42, March 2007.

[8] A. Kantee. Rump file systems: kernel code reborn. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 15–15, Berkeley, CA, USA, 2009. USENIX Association.

[9] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[10] A. Tanenbaum, J. Herder, and H. Bos. Can we make operating systems reliable and secure? *Computer*, 39(5):44 – 51, may 2006.

[11] A. S. Tanenbaum, H. van Staveren, E. G. Keizer, and J. W. Stevenson. A practical tool kit for making portable compilers. *Communications of the ACM*, 26:654–660, September 1983.

[12] The GNU Project. GCC, the GNU Compiler Collection, version 4.4.3. `http://www.gnu.org/software/gcc`.

[13] The GNU Project. GNU binutils, version 2.17. `http://www.gnu.org/software/binutils`.