Early FreeBSD Ports

BY DOUG RABSON

From its inception, FreeBSD focused on providing solid support for the i386 architecture. The PC platform was commonly available and relatively inexpensive and concentrating our existing resources on this helped to make FreeBSD on i386 stable and performant. However, after a few years, we decided to broaden our support; the first target was the DEC Alpha platform which was a good choice given its 64-bit architecture. A few years later, we added IA-64 support, then being positioned by Intel as a successor to the i386.

FreeBSD on Alpha

In early May of 1997, Jordan Hubbard asked for volunteers to work on porting FreeBSD to the DEC Alpha platform. The Alpha was a 64-bit, load/store architecture, which was quite different to i386 with more registers and a RISC instruction set. Available hardware used the familiar PCI and ISA bus interfaces.

I volunteered for the project. Clem Cole at DEC loaned us some hardware which arrived in July 1997 (apart from Peter Wemm's machine which I seem to remember got lost in customs). At this point, I think it started to sink in how much work this project was going to take, with changes needed in many different areas.

Device drivers in FreeBSD, especially for ISA, were going to need changes to support the new architecture where hardware access was quite different.

Drivers

Device drivers in FreeBSD, especially for ISA, were going to need changes to support the new architecture where hardware access was quite different. To make this work and share as much code as possible between i386 and Alpha, we needed an abstraction layer.

I had some ideas for this which eventually turned into FreeBSD's newbus framework. My plan for this was to discover devices automatically, starting from the top-level system device bus which was typically PCI and ISA for both i386 and Alpha. The kernel would dynamically build a tree of devices and then match these with available drivers. I also wanted to be able to combine this with earlier work on the Kernel Linker (KLD) to allow drivers to be added after the initial boot. I worked on a prototype for this in late 1997 but didn't get as far as supporting any 'real' hardware drivers.

File Formats

In 1997, FreeBSD was using the a.out file format which was a very simple 32-bit format. Most other Unix-like systems were using ELF format by this time which provided more flexibility and already supported 64-bit platforms including Alpha.

Moving to the new format involved changes in the build system to support the new format and also to support dynamic linking, which is implemented in userland using the Runtime Linker (RTLD). Lots of people worked on this including John Polstra, Jordan Hubbard, Peter Wemm and myself over several months in early 1998.

Booting

This was initially an area of uncertainty for the project. The Alpha 'pre-boot' environment was fragmented with Digital Unix (DUX) and VMS using the SRM Console, NT using AlphaBIOS and Linux using both of these as well as their own MILO.

Any operating system on Alpha needs something called PALcode to handle virtual page translation, caches, interrupts and transitions between user-mode and kernel-mode. There were variants of PALcode for DUX, NT and VMS. While the DUX PALcode was likely to be our best choice, it wasn't clear if we could use it without an expensive licence. As it turned out, all the Alpha hardware we ended up supporting had support for the SRM Console which came with DUX PALcode, so this ceased to be a problem.

The i386 boot code was limited in size to just 7.5k. The limit came from the UFS filesystem format which reserves an 8k area for the bootstrap, and we needed a 512 byte sector from that to allow the PC BIOS to boot. This was just enough to read the a.out format kernel file from the root filesystem into memory and start it.

Alpha would have a similar limit; the SRM Console used the first sector of the disk to identify a contiguous range of sectors to load, which left 7.5k from the UFS boot area in the same way as i386. On Alpha, the 7.5k limit probably wasn't going to be enough due to the lower code density of the RISC architecture and the extra complexity needed for the ELF format. We ended up rewriting the bootstrap so that the 7.5k boot stage loaded a larger boot program (in modern FreeBSD systems this is /boot/loader). This gave us enough flexibility to fully support booting ELF format kernels on both i386 and Alpha as well as other new features such as pre-loading kernel modules, network booting, and more. Mike Smith worked on the multi-stage boot and Peter Wemm implemented module pre-loading. Somewhere along the way, a Forth interpreter was added—I think Jordan Hubbard was responsible for that.

Userland

When we started this project, the FreeBSD source tree was not set up for easy cross-compiling. This made building user-mode utilities a little challenging. John Birrell worked on getting most of the FreeBSD source tree to build on a NetBSD host and had a system with a fairly complete FreeBSD userland running with a NetBSD kernel.

The NetBSD system call interface was a little different from FreeBSD, so John's early userland work used NetBSD's ABI. A native FreeBSD kernel would need utilities that used the FreeBSD

ABI, but this was a significant step forward. Once we had a working kernel, it was a relatively straightforward process to move from the hybrid NetBSD/FreeBSD system to a working native FreeBSD system.

Kernel

This was probably the largest part of the project and involved filling in all the machine-dependent parts of the kernel which provide low-level support for virtual memory, interrupt handling, process context switching etc.

I approached this by building an Alpha cross compiler and just attempting to build a kernel, seeing what failed to compile, then filling in the gaps, either with empty stubs or by importing code from NetBSD where it was similar enough to work for FreeBSD. This was a fairly tedious process which took several days, but eventually ended with a non-functional kernel binary.

The next, longer, part of the port was to attempt to run this kernel, seeing how far it got until something broke and then fix that problem before trying again. To run each test, I used a tool called SimOS. This simulated an Alpha-based computer complete with simulated hardware such as disks and serial ports. SimOS supported debugging the simulated kernel with gdb; this was extremely helpful since I was able to single step through the very early kernel initialization process which sets up the kernel virtual memory, etc. before moving onto the machine-independent initialization sequence.

To shorten the porting process, I used code from NetBSD/alpha where it made sense. Unfortunately, I omitted the NetBSD copyright in a few places. This had to be fixed in public after the code was committed which was quite embarrassing. This is one of the very few times where FreeBSD's commit history was altered—we removed the revisions with incorrect copyrights.

One area where using NetBSD code wasn't going to work was in the virtual memory support where FreeBSD was quite different. The Alpha page tables were similar to i386 with a tree-based structure using three levels (where i386 used two levels at the time). I copied the i386 code and changed it to add the extra level.

Initial support for Alpha was committed in July 1998 with support for the SimOS emulator and real hardware support followed over the next few months. The release notes for FreeBSD 3.0 mention this: 'A port to the DEC Alpha architecture has entered "ALPHA" (haha) status'.

FreeBSD on IA-64

This project got started in 2000 when Paul Saab brought a set of IA-64 documentation to Usenix ATC and asked me if I would be interested in porting to this new platform. At the time, Yahoo! was a large-scale user of FreeBSD on i386 and some of their workloads were running up against limitations of the 32-bit platform.

The IA-64 architecture was interesting in several ways. The instruction encoding was in 128-bit instruction bundles, each of which contained up to three 41-bit instructions which may execute simultaneously. This allowed for a large register set with 128 general purpose registers and 128 floating point registers.

The general-purpose registers are divided into two groups--32 'static' registers and 96 'stacked' registers which the processor would allocate from a large pool of registers and automatically save and restore on function call and return. Each register is 64 bits plus one NaT ('Not a Thing') bit used for speculative execution.

Conditional execution is via 64 predicate registers which are each a single bit and hold the result of compare instructions. Each instruction can be conditionally executed based on the value of a predicate register.

Indirect branches (e.g., function pointers) are supported using 8 branch registers which can help with branch prediction.

Virtual memory management is controlled by a Virtual Hash Page Table (VHPT) which contains a subset of possible virtual to physical mappings. A software TLB miss handler is used to find translations which are not in the VHPT. The VHPT supported two formats, a 'short' format which could be used to emulate traditional tree-based page tables or a 'long' format which was a simple hash table with collision chains.

Booting

The IA-64 hardware used the EFI pre-boot environment. I added very basic support for EFI to the multi-stage boot loader. In the IA-64 EFI environment, programs were relocatable; this needed to be done in the EFI program itself which was difficult to debug.

Initial support for Alpha was committed in July 1998 with support for the SimOS emulator and real hardware support followed over the next few months.

Userland

Porting the FreeBSD user-space tools and utilities was fairly straightforward—the FreeBSD build supported cross-compiling by this time and only needed the addition of IA-64 versions of low-level library code for things like string comparison, memory copy, and system calls.

Kernel

We were lucky enough to have access to an HP IA-64 emulator (SKI) from which Marcel Moolenaar made a FreeBSD port. This included an instruction-level debugger which was very helpful in debugging early kernel initialisation and trap handling.

The kernel port was a little more difficult than Alpha. This time, there wasn't another BSD port which could be used for reference, so all the low-level support was new code. The IA-64 architecture required two stacks, one for registers and one for regular data. Trap handling was significantly more difficult than most other architectures due to the extra register state and the complexity of speculative execution and the stacked registers.

The long-form VHPT format ended up being a reasonable fit for FreeBSD's virtual memory system. The machine-independent VM system makes requests to the platform's pmap system to make virtual to physical mappings. These were just added to the VHPT.

32 Bit Compatibility

During development of the port, we used Perforce for source code control and there was only an i386 binary available at the

time. I wanted to be able to use this on the target platform during the port, so I ended up implementing i386 compatibility which used the built-in i386 support in the IA-64 processor. This built on earlier work on Linux and SVR4 emulation which had made a clear separation between the syscall ABI and implementation.

Legacy

The Alpha port prompted a great deal of necessary supporting development which has helped to shape the modern FreeBSD kernel. The transition from a.out to ELF format was a necessary step for the Alpha port, but since ELF rapidly became the de-facto standard, moving away from a.out on all platforms saved us from having to spend large amounts of effort supporting and extending an obsolete format. The multi-stage boot loader has proven to be a flexible platform, making new architecture ports easier and supporting booting from modern file systems such as OpenZFS. The newbus device framework facilitates driver compatibility across architectures and supports dynamic device discovery which is required in modern systems where devices can be added or removed at any time.

Adding support for Alpha forced us to tackle 64-bit compatibility problems across both kernel and user. The load/store architecture uncovered other problems such as the assumption that read-modify-write operations on memory to set flags or increment counters could not be affected by hardware interrupts. This was solved by adding a set of 'atomic' operations to the kernel. The atomics framework was extended by John Baldwin to support IA-64's acquire/release semantics and is used extensively to support multi-cpu platforms.

The IA-64 port was inspired by the need to get past the limitations of the 32-bit i386 platform while retaining compatibility with legacy software. While these goals were achieved, the platform itself did not reach the price/performance of the simpler i386 architecture. IA-64 eventually found its niche in large-scale Supercomputing, but it was not a good fit for most FreeBSD workloads and was superseded by AMD's x86-64 extension to the i386 architecture, which is pervasive in modern compute environments.

Support for both platforms has since been removed from FreeBSD. The Alpha architecture was a casualty of the Digital/Compaq merger, although it continued to be available as a product until 2007. FreeBSD support was removed in 2006. Support for IA-64 survived a little longer; Marcel Moolenaar made many improvements over the years to support multi-processor and NUMA variants of the platform. Support was removed from FreeBSD in 2014 and the platform was discontinued in 2021.

DOUG RABSON is a Software Engineer with more than thirty years of experience ranging from 8-bit text adventure games back in the 1980s to terabyte-per-second distributed log aggregation systems in the 2020s. He has been a FreeBSD project member and committer since 1994 and is currently working on improving FreeBSD support for modern container orchestration systems such as podman and kubernetes.