Bootstrapping Plan 9 on PCs

Geoff Collyer
geoff@plan9.bell-labs.com

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

What's interesting or tricky about bootstrapping Plan 9 on PCs?

Introduction

Plan 9 has new PC bootstraps, *9boot* and *9load,* replacing the decade-old *9pxeload* and *9load* programs. What did we learn while writing them?

PC Constraints

The IBM PC imposes quite a few constraints on bootstrap programs (programs that load operating system kernels). A PC starts executing in 16-bit 'real' (Intel 8086) mode and has no boot monitor, as other machines do, just a primitive BIOS that will perform a power-on self-test (POST) and attempt to read boot sectors from disks or load a modest payload from the network via TFTP. (Actually some new machines have slightly less primitive boot loaders called (U)EFI, but we don't deal with EFI.) The boot sectors must load further bootstrap programs that resemble the TFTP payload. These bootstrap programs can only address the first megabyte of memory until they get out of real mode, and even then the upper 384KB of the initial megabyte is reserved for device ROMs.

BIOS calls (via the `INT` instruction) only work in real mode, so the bootstraps execute BIOS calls to learn the machine's memory map and power management configuration, and stash the results in the first megabyte for later retrieval by the loaded kernel. Empirically, some BIOSes enable interrupts (with `STI` instructions) during BIOS calls, so the bootstraps disable them again after each call; failure to do so often results in an interrupt, perhaps from the clock, resetting the machine. `9loadusb` returns briefly to real mode to read USB devices and has mixed results with that.

Getting into 32-bit protected mode permits addressing the first 4GB of memory, but first it is necessary to enable the A20 address line (the `1<<20` bit). For (extreme) backward compatibility, this bit is normally held to zero until software requests that it be released, and holding it to zero will cause references to the second megabyte to be mapped to the first, etc., causing bizarre-seeming memory corruption. The old technique was to ask the keyboard controller to release it, but some systems now have no keyboard controller (they are servers or have USB keyboards). We have found it necessary to keep trying different methods until one succeeds and is verified to have succeeded. The new bootstraps also try an `INT 15` BIOS call and manipulation of port `0x92` ('system control' on some systems).

Even in protected mode with A20 enabled, some systems force a gap in the physical address space between 15MB and 16MB, which must be avoided.

Plan 9 Requirements

- The new bootstraps must be able to load 64-bit `amd64` kernels as well as `386` ones. In addition to Plan 9 boot image format, the new bootstraps understand ELF and ELF64 formats.
- Plan 9 kernels need to be started in 32-bit protected mode and implicitly assume that A20 is enabled.
- They expect a parsed `/cfg/pxe/`*ether* or `plan9.ini` file to be present at physical address `0x1200` and that *9load* will have added entries to it describing any disk partitions found. (*9boot* does not do this and so kernels loaded by it that care about disk partitions will need `readparts=` in `plan9.ini`.)
- They expect automatic power management information obtained from the BIOS to be present in the first megabyte.
- Our `amd64` kernels (9k, Nix, etc.) also expect a Gnu *multiboot* header containing any arguments and a memory map.

Non-Requirements

- The bootstraps should ignore secondary processors, leaving them in reset.
- The bootstraps need not do anything with floating point.

Techniques and Tricks

Our new bootstraps are stripped-down Plan 9 PC kernels without system calls and user mode processes. They share the vast majority of their code with the ordinary PC kernels; about 10,000 lines of C are new or different in the bootstraps. In particular, they use the ordinary PC kernel's device drivers, unmodified. *9boot* loads kernels via PXE; *9load* loads kernels from disk. This is more specialised than the old all-in-one *9load*.

From protected mode, the bootstraps initially enable paging for their own use. Before jumping to the loaded kernel, they revert to 32-bit protected mode, providing a known initial CPU state for the kernels.

Self-decompression of the bootstraps can help to relieve the 512KB/640KB payload limit. Russ Cox's decompressing header* code is about 9K all told, including BIOS calls to get APM and E820 memory map info. We only bother with this currently for *9boot*. The bootstraps also will decompress *gzip*-ped kernels loaded from disk, mainly for CD or floppy booting, where they are limits on kernel size.

Figure 1 shows the memory map in effect while the bootstraps run.

Our USB stack (at least 3 HCI drivers plus user-mode drivers, implying system calls and user-mode support) is too big to fit in the first 640KB, so the bootstraps try to get BIOSes to read from USB devices and some of them do.

We strongly prefer PXE booting; disk booting is a poor second. PXE booting minimises the number of copies of kernels that must be updated and ensures that machines boot the latest kernels. Reading via 9P (as user *none*) would be even better: just read `/cfg/pxe/$ether` and `/386/9pccpu`. This would probably require adding `devmnt` back into the bootstrap kernels.

---
* see `http://plan9.bell-labs.com/wiki/plan9/Replacing_9load`

Figure 1: Layout of physical memory during bootstrapping

0	misc., including bios data area
31K	start of pxe decomp + compressed 9boot. decompresses to 9MB.
64K	start of pbs
512K	pxe loader from ROM
640K	UMB; device ROMs
1M	kernel
9M	9boot after decomp. (decompresses kernel.gz at 13M.) loads kernel at 1M.
13M	(kernel.gz)
15M	no−man's land
16M	malloc arena for 9boot
...	

Future H~~orrors~~ Directions

We haven't dealt at all with (U)EFI, 'secure boot', GPTs nor GUIDs. We can use Plan 9 partition tables instead of GPTs to address disks larger than 2 TB.

Lessons Learned

A disabled A20 line can masquerade as all sorts of baffling problems. It is well worth ensuring that it is truly enabled.

Virtual−machine hypervisors can be good test−beds and provide better crash diagnostics than the blank screen you get on real hardware, but they can also mislead (e.g., `amd64` kernels on Virtualbox, Vmware 7 on Ubuntu 12.04).

All of these bootstrap programs and the BIOS (and POST) can be avoided, once Plan 9 is running, by using `/dev/reboot` as packaged up in *fshalt*(8), which is much faster.