Automatic Boot Configuration

Erik Quanstrom
quanstro@quanstro.net

ABSTRACT

Boostrapping a kernel is a tedious and error-prone process. Ideally it should "just work." But in practice this has been hard. Often there are many interfaces or drives to choose between. And one must carefully arrange agreement between the loader and kernel. One simplifying assumption might be to have the loader and kernel follow the firmware's (BIOS's) lead, and use the device the firmware used to load the loader. We explore a strategy for getting this information from PC BIOS, and the results.

Introduction

I just hate to be pushed around by some ... machine.
— Ken Thompson, on the i960

In our environment, we PXE almost all machines. And this has been a great time saver. Part of this convenience has been accidental. For many years, even if our machines had two interfaces, they were the same sort of gigabit interface. BIOS kindly arranged them so the bottom left one would be discovered first if we just followed our noses during device enumeration. However as we needed more bandwidth, and began to add additional gigabit, and then 10gbe (10 gigabit Ethernet) cards, our odds of accidental success went down. PXE failure became rather common.

The first attempt working around this was by careful ordering of ethernet devices. This took two forms. First, the different drivers needed a consistent ordering. This comment is from a recent sources kernel and illustrates the issue

```
# order of ethernet drivers should match that in ../pcboot/boot so that
# devices are detected in the same order by bootstraps and kernels
# and thus given the same controller numbers.
```

Second, even within a driver, more specificity was required. Most machines we have ship with Intel PCIe Ethernet chipsets. Plan 9 supports 16 major variants. Some are exclusively used for LOM (LAN on motherboard), and one can improve the odds of booting correctly by calling each variant out as a separate Ethernet type, the carefully ordering them. Sometimes this alone is enough. Other machines needed simply

```
ether0=type=i82572
```

to disambiguate.

The problem was this wasn't enough. Before **plan9.ini** can be loaded, the interfaces need to be enumerated. This means that to load the file that gives us the network device ordering, we first must order the network devices. Working around this issue

often required plugging in extra cables to satisfy the loaders misguided view of the world. Worse, this confuses ARP tables and networking equipment. Many machines just refuse to boot.

Built In Configuration

The first attempt to address this situation was building the configuration file into the loader. A prototype was written to help a few problematic machines boot. The prototype required that each machine configuration (even differences unrelated to the ethernet) have its own loader. A **mkfile** automated much of this. But nonetheless it was not a satisfying solution.

BIOS To The Rescue

Never hold discussions with the monkey when the organ grinder is in the room.
— Winston Churchill

The BIOS knows what hardware it is using, otherwise it couldn't start the PXE process to begin with. If the loader interacted with BIOS directly, the PXE calls UNDI GET NIC TYPE and UNDI GET INFORMATION can give us information on the PCI TBDF (type, bus, device function), and station address of the active nic. These could be used by the kernel to identify the first ethernet port. Then we would be guaranteed to select the same interface for ether0 that BIOS PXE'd from.

The first challenge in making this work is talking to BIOS. Adding BIOS support to *9load* has been a challenge. Driving some hardware directly, and then switching back to BIOS for other hardware has proved problematic. We do however have *iplpxe* (a descendant of cinap_lenrek's *9boot*) which interacts with the hardware using only BIOS calls. *Iplpxe* is only 1876 lines of stand-alone code. Its executable is 9292 bytes. For comparison, *9boot* is 13701 lines of code, plus libraries and portable code. Its executable is 405285 bytes.

The aforementioned PXE calls were added to *iplpxe* to collect information, and whenever there is no **ether0=** configuration in the **plan9.ini**, *iplpxe* adds one of the form

```
ether0=ea=002590c1a521 tbdf=0x0c020100
```

In the PC kernels the test for matching port (address) in the ethernet drivers was replaced with a call to *ethercfgmatch*(edev, pcidev, port) which uses the TBDF to detect the matching hardware.

While simple, this approach has been effective. No current CPU servers need to have manual Ethernet configuration.

Booting From Disk

When booting from disk or USB disk, the problems are similar. Which target on which of the myriad of controllers was the boot device? Fortunately, T13 BIOS Enhanced Disk Drive[2] can also provide the TBDF and target number of the BIOS boot drive (0x80). The FAT and ISO boot loaders, *iplfat* and *ipliso*, set the boot variable **drive0** if not already set. For example with a SATA or IDE drive,

```
drive0=sectors=0x000000003a386030 tbdf=0xc00fa00 chan=0x01
```

would represent a PCI device on 0.31.2 on the second controller port (or slave). An interesting special case is booting from USB devices. For example,

```
drive0=sectors=0x0000000000ee8c00 tbdf=0xc00d000 usb=y chan=...
```

In theory **chan** should be the WWN of the USB disk device, but in this case that appears not to be the case. It is not yet clear if this is a bug in *iplfat* or if one cannot depend on this value being set correctly. To work around this, the current code assumes that there is exactly one USB disk at boot time. Regardless, the kernel boot code recognizes USB devices and uses the *sdloop*(3) to mount the USB device as the *sd*(3) device **sdu0**. This allows it to be partitioned. At the cost of about 20 lines of code, this allows simple booting from USB disks with no other media. For comparison, the sources version is 153 lines.

Conclusion

Using BIOS services allows for accurate identification of BIOS device ordering while at the same time expanding Plan 9's ability to boot from arbitrary PC devices and reducing the amount of code devoted to bootstrapping.

Abbreviated References

[1] Preboot Execution Environment (PXE) Specification, version 2.1, Sept 20, 1999

[2] BIOS Enhanced Disk Drive Services 3 (EDD-3), T13, Nov 5, 2004.