A new boot process for Plan 9 from Bell Labs

Iruatã Souza
iru.muzgo@gmail.com

ABSTRACT

We describe a new way of booting Plan 9 in which no special bootstrap program is involved; rather, the bootstraping is done by just another Plan 9 kernel. In this way, we can take advantage of the full range of devices, networks, and file systems supported by the operating system.

1. Motivation

The Plan 9 4th Edition boot process has been the target of critics from the Plan 9 users and its replacement has been discussed for years [1]. We tried to synthesize these discussions and ideas in the form of a rewrite of the relevant boot routines.

2. Introduction

Plan 9 boots on PCs with help from an auxiliary kernel called *9load(8).* Such a kernel does not pretend to be general on its purpose and does not share its source code tree with the other Plan 9 kernels. This approach has shown limitations and in order to have these limitations addressed, fundamental changes needed to be made; the current status and future directions of the new Plan 9 boot process is hereafter discussed.

3. Plan 9 on PC

In order for this paper to be self-contained, this section presents the minimum required information about Plan 9 running on computers of the x86 architecture.

3.1. Storage

We assume the storage medium to be some form of local disk. Such a disk can be divided into slices/partitions, in which case it is said to be partitioned.

The slice reserved for Plan 9 (or the whole unpartitioned disk) is itself divided into Plan 9 partitions. A standard Plan 9 installation taking the whole disk and using *fossil(4)* as the root file sytem layouts the disk as follows (Begin and End in sector units):

The code implementing the solution was initially written as part of the *Google Summer of Code* 2009 program.

Name	Start	End	Description
9fat	63	−	Plan 9 kernels and boot configuration
nvram	−	−	non−volatile ram for PCs
fossil	−	−	*fossil(4)*
swap	−	−	swap area

3.2. Boot

After the Power On Self Test, the BIOS loads sector zero of the disk into physical memory address 0x7C00 and jumps to that location. If the disk is partitioned, that sector will contain the Master Boot Record (MBR). The MBR searches the master partition table for the active partition, loads that partition's Partition Boot Sector (PBS) into 0x7C00, and jumps there. If the disk is not partitioned, sector zero is the PBS itself.

3.3. PBS

The partition boot sector starts by jumping over its Boot Parameter Block (BPB) [2]. Its BPB ID field is edited by *format(8)* to contain the starting sector of the root directory of the FAT filesystem to where it is being installed. That directory is searched for a file named '9LOAD   '. If the desired file is found, the PBS calls BIOS interrupt 13 [4] to read the file's contents to memory 0x1000 physical and far jumps there.

PBS is hardcoded as a FAT boot sector and only understands FAT filesystems. Then it can only be used if a FAT is present at the beginning of the Plan 9 disk slice. Because of the x86 segmentation used by the PBS and the BIOS interrupt, *9load(8)* has a size limit of approximately 1MB.

3.4. 9load(8)

9load(8) is the PC bootstrap program: an auxiliary kernel with its own source code tree. Its main purpose is running a Plan 9 kernel. For this purpose to be met, *9load(8)* needs to a) enable 32-bit protected mode, b) load boot configuration, and c) find and load a kernel.

Plan9.ini(8) is the boot configuration file for PCs. *9load(8)* probes storage media searching for files plan9.ini and plan9/plan9.ini. When a file is found, it reads at most 100 configuration lines in the form name=value, storing them in memory at CONFADDR (0x1200) in order for the loaded kernel to read.

The bootfile line in *plan9.ini(8)* may contain the kernel path; if no such line is found, a prompt is presented for the user to type the desired path. The kernel must be either in *a.out(6)* or ELF format and can be gzip compressed. *9load(8)* reads the kernel's text segment to virtual 0xF0100000 and the data segment to the first page-aligned address after the text segment's end. Everything in place, *9load(8)* jumps to the kernel entry point at virtual 0xF0100020.

The way *9load(8)* works shows some limitations. Except for bootfile, the only way to set boot configuration is by using *plan9.ini(8).* We recognize that having a permanent configuration is a valuable feature, but forcing the user to edit a file everytime she wants to experiment with boot parameters does not seem to be optimal. In addition, the bootstrap program requires *plan9.ini(8)* and the kernel to be in a FAT partition.

A minimal device driver and filesystem infrastructure is needed for *9load(8)* to do its job. Since its source code is not the same as the other kernels, the exiting structure of the later can not be enjoyed by the former. This leds to duplicated effort, where drivers need be ported from kernels to *9load(8)* if kernel supported hardware is to be used in the boot process.

3.5. Stock Kernels

Plan 9 kernels assume to some degree that they are bootstrapped by *9load(8).* In any setup, at least 32-bit protected mode is expected to be turned on in the processor.

If *draw(3)* is going to be used, the kernel will rely on *9load(8)* to have setup part of VGA configuration; and if APM is needed, again the kernel will take it for granted from the bootstrap program. Even that *sd(3)* needs to have an in-memory table of partitions it does not parse disks for that information, expecting that *9load(8)* has done the parsing and stored the table in a CONFADDR line.

3.6. boot(8)

Besides being linked in the kernel image itself, *boot(8)* is mounted at /boot/boot and is the first user program to run. It connects to the file server specified by the user (via *plan9.ini(8)* or prompt) and mounts it as the namespace root. It then spawns a new process and run *init(8).*

Boot(8) has limitations akin to *9load(8)'s.* It does not fully enjoy the features provided by the system. As an example, the namespace root, if local, must be *kfs(4)* or *fossil(4);* if another filesystem is needed, routines particular to *boot(8)* must be written even if the supporting programs already exists in Plan 9.

4. 9null

In order to address the mentioned limitations, we wrote a new PBS, modified *boot(8)* and created a new kernel configuration. To the effort under which this work was done we gave the name 9null.

Initially the effort was only to remove *9load(8)* and boot a kernel directly. Russ Cox did solve part of the problem by writing a minimal bootstrap program that would load a kernel linked with itself [1]. His solution still left us a) the need for *plan9.ini(8),* b) the need for the kernel and *plan9.ini(8)* to be on 9fat, and c) the need for the local root to be either *kfs(4)* or *fossil(4).* We solved a) and c) with modifications to *boot(8),* and b) with a new PBS.

4.1. pbs32.s

Our new PBS is pbs32.s. Its first task is to make the switch to 32-bit protected mode so that it can address the whole address space. As a side-effect, the kernel size limit which existed in the old PBS vanishes.

The sectors of the Plan9 slice on disk are layouted as follows:

```
Sector Description
0      Partition Boot Sector
1      Plan 9 partition table
2..k   Space reserved for kernel (and possibly configuration)
k..n   data
```

Pbs32.s parses the master partition table, if any, to find the first sector of the Plan 9 slice; if the table is empty, Plan 9 sector zero coincides with sector 0 of the disk. The PBS then loops using ATA commands to read disk sectors and checking them for the *a.out(6)* signature; if such a sector is found, the file is read to physical memory 0x00100000 with proper alignment guaranteed. At last, a jump is made to the kernel entry point at 0x00100020 physical.

Such approach poses the need for the kernel to be contiguously placed on disk. On the other hand, since it only knows about disk sectors, it is filesystem agnostic and the space reserved for kernel storage does not even need to have a filesystem at all.

4.2. 9pcload

9pcload is the kernel created to be loaded directly by pbs32.s. It consists of minor additions to the pcf configuration so as to help the modifications made in *boot(8).* The biggest difference is that *rc(1)* was added to /boot.

4.3. Boot(8) modifications

Boot(8) was modified in three places. 9pcload, in cooperation with /sys/src/9/boot/mkboot, sets a global variable pcload to 1 that allows *boot(8)* to know if it is booted by pcload.

In the case 9pcload started *boot(8), plan9.ini(8)* is scanned, loaded, and the configuration added to #ec for the next kernel to use; *boot(8)* asks for a kernel to be booted – instead of the root file server – and use *reboot(8)* to reboot into it. If '!' is given as the kernel, *rc(1)* is started and the boot process may be carried manually. For the other kernels, *boot(8)* behaves as described in 3.6.

5. Open questions and work in progress

In order for 9null to fully replace the current boot process, there are some questions that need to be addressed and work that need to be finished.

Pbs32.s uses ATA commands to read sectors from disk, so using it on floopies is not supported. Since a great part of the machines comes with no floppy drive anymore, that does not seem to be a real problem. In any case, the old PBS can still be used when booting from floppy.

The new PBS does not support compressed kernels, though this only seems to be a problem when booting from floppies. Since those are not supported, it can be regarded as a non-issue.

Boot(8) is being rewritten to be minimal enough so as to let *rc(1)* carry the boot process. This raises the possibility for *boot(8)* to enjoy the full range of Plan 9 services.

At last, we have not put the effort to solve the problem of PXE booting but understand that, in order to preserve Plan 9 principles, methods of booting remotely must be supported. We believe that it is even paramount that the new boot process be as agnostic as possible to the location of its kernel and root.

6. Conclusions

We have managed to write a new boot process for Plan 9 in which a kernel is directly loaded by the early, sometimes called first stage, bootloader. Even not ready for production use yet, it shows advantages over the old boot process, being at the same time simpler and more general.

Our hope then is that this work will allow more users to boot Plan 9.

7. References

[1] R. Cox, *9fans archives*, `http://9fans.net/archive/2005/12/81`, 2005

[2] Microsoft® Corporation, *Microsoft Extensible Firmware Initiative FAT32 File System Specification*, `http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx`, Version 1.03, December 6, 2000

[3] *Plan 9 Manuals*, current edition published online at `http://plan9.bell-labs.com/sys/man`

[4] Intel® *Intel® 64 and IA-32 Architectures Software Developer's Manual*, Volume 3A: System Programming Guide, Part 1 and Volume 3B: System Programming Guide, Part 2