Sunrise
DRAFT

Anthony Sorace
a.9srv.net

Strand 1 Technologies

ABSTRACT

DRAFT Our lab includes a variety of older Sun hardware. This describes how we boot some of it.

1. Introduction

Our lab has a collection of older Sun systems which we would like to use. This set currently includes:

sysname	model	RAM
arabica	JavaStation	
kona	JavaStation	
bluemountain	JavaStation	
solar	Ultra 5	
vega	SPARCstation 20 MP	256 MB
aldebaran	SPARCstation 2	64 MB

All of these systems lack integrated storage: the Ultra 5 has a bad disk, the SPARCstations were obtained without hard disks, and the JavaStations have no capability for internal storage. I would like to be able to boot each of these over the network and run different systems on each. This era of Sun systems have good capabilities for network booting, although the procedure differs quite a bit from more modern systems.

2. Prepare a Plan 9 Server

We use Plan 9[0] to configure the Sun systems and respond to their requests over the network. To do this, your Plan 9 server should be running *ip/dhcpd*, *ip/tftpd*, and *ip/rarpd* on the interface connected to the network the Sun is on. In our lab, we include these lines in the startup for the Raspberry pi used as a boot server:

```
ip/dhcpd
ip/tftpd
ip/rarpd -e /net/ether1
```

The server also needs an appropriate entry in your *ndb* database. A representative example from our lab:

[0] The procedure described has been tested with 9legacy and Plan 9 4th Edition. Other distributions should likely work but are untested.

```
ip=10.1.20.153 sys=vega ether=080020714198
        bootf=/sparc/9ss10
        auth=10.1.20.120
        cpu=10.1.20.120
        fs=10.1.20.120
```

The file named on the *bootf* line must exist and be readable. The *auth*, *cpu*, and *fs* lines are not strictly needed for netbooting the Sun, but will be important for booting Plan 9, described later.

If you're using a stock Plan 9 system, you'll need to fix a bug in *tftpd*. In */sys/src/cmd/ip/tftpd.c*, change the line

```
if(suffix-name != 8 || (strcmp(suffix, "") != 0 && strcmp(suffix, ".SUN") != 0))
```

to

```
if(suffix-name != 8 || (strcmp(suffix, "") != 0 && strncmp(suffix, ".SUN", 4) != 0))
```

and recompile tftpd. See *Loading a Kernel*, below, for more information.

You may need additional services if you're booting Plan 9 on the target (in addition to using it as a boot server). Those requirements are described in a separate section, below.

3. Serial Communication

While the objective is to get the Sun systems booting over the network, initial configuration will be done over the serial line. It is usually possible to do this configuration via an attached keyboard and monitor, but a serial port is usually easier to script, which is especially helpful on older systems (see the section on NVRAM, below). We have two methods of connecting a Raspberry pi to the Sun serial port.

3.1. USB Serial

The easiest way to get a serial connection on a Raspberry Pi is with a compatible usb-to-serial cable. Plan 9's `usb/serial` supports FTDI adaptors and cables. Compatiblity is hard to guarantee ahead of time, but `/dev/usb/ctl` reports the cable we have as:

```
255 csp 0xffffff vid 0x0403 did 0x6001 FTDI 'FT232R USB UART' xhci
```

It is branded "TERA", although we cannot verify its source. When using a device like this, `usb/usbd` (which should already be running) will configure the device as `/dev/eiaU0/eiaU` (with a matching *ctl* file). Use those file names in the configuration below.

3.2. TTL Serial

The other common alternative for getting a serial port on a Raspberry pi is to use the built-in TTL serial device via a ttl-to-rs232 converter connected to the GPIO pins. One procedure for doing this is described at Setting up RS-232 on a Raspberry Pi. The procedure described there initially used the pi's *uartmini* driver, which controls an older, simpler, and less full-featured uart on the Raspberry Pi. With this uart, I was unable to find a configuration the driver supported which proved "safe" for the automated configuration program (*bootscript*, described below). Instead, a new program, *trickle*, was written to simply copy the uart's input and output at a slower rate. This was effective, but complicates the procedure.

Instead, I'm now using the `pl011` driver, added to the Raspberry Pi kernel in 2023. With that driver, enabling the device FIFOs at a high level seems to be sufficent. To set the FIFOs to their maximum level and flush any data in the queue, run:

```
echo i99 > /dev/eia2ctl
echo f > /dev/eia2ctl
echo h > /dev/eia2ctl
```

(replacing `/dev/eia2ctl` as appropriate).

In the `config.txt` file used for booting your Raspberry pi, you'll need to ensure the line `enable_uart=1` exists, in either the `[all]` block or the appropriate block for your model (if you are using blocks in that file).

3.3. Configuring the Port

All of these Sun systems come up with their serial port configured to communicate at 9600 baud with 8 data bits, no parity, and 1 stop bit (often written `9600,8,n,1`). Serial ports on Plan 9 usually come up with 8 data bits, no parity, and 1 stop bit, but the baud rate can vary. Set it before proceeding with `echo b9600 > /dev/eia2ctl`, (replacing `/dev/eia2ctl` with the appropriate control file for your serial console). It is also worth confiming the other three settings, especially if using a usb device; see *uart(3)*.

Note that SPARCstations and similar Sun systems typically pick their output device based on whether or not a keyboard is attached. To get the Sun to use the serial port as its console, ensure no keyboard is attached at boot time.

4. Configure NVRAM

All of these systems begin their netboot process using *bootp* to obtain an IP address (they will also use the address of the responding server later). This relies on some information stored in NVRAM, but all of this hardware is old enough that the NVRAM battery is dead. Unfortunately, this era of Sun hardware all used NVRAM chips where the battery is integrated; replacing the battery involves some degree of surgery on the chip itself. The configuration stored by the NVRAM can be entered at the console, but will be lost upon each power cycle[1]. To avoid manually re-entering this information on each boot, a new script, *bootscript*, provides it. Invoked as `bootscript <sysname> <device>`, it will look up the IP address for *sysname* in *ndb*, attach to a serial console on *device*, and prompt the user to power on the system. It will then monitor the output from the console and provide the needed settings.

In the representative session below, `:;` is the prompt text following that is user input.

[1] Settings will sometimes be preserved across very short power off/on cycles, but the period is too unpredictable and too short to be useful.

```
:; cat /dev/eia2status
b28800 c0 d1 e0 l8 m0 pn r1 s1 i0
dev(2) type(0) framing(13637) overruns(0) berr(0) serr(0) cts
:; echo i99 > /dev/eia2ctl
:; :; echo f > /dev/eia2ctl
:; echo k > /dev/eia2ctl
:; echo h > /dev/eia2ctl
:; echo b9600 > /dev/eia2ctl
:; cat /dev/eia2status
b9600 c0 d1 e0 l8 m0 pn r0 s1 i99
dev(2) type(0) framing(22070) overruns(0) berr(0) serr(0) cts
:; bootscript arabica /dev/eia2
08 00 20 c0 ff ee c0ffee mkpl
Power on arabica now.
Starting real time clock...
Incorrect configuration checksum;
Setting NVRAM parameters to default values.
Setting diag-switch? NVRAM parameter to true
Probing CPU FMI,MB86904

The IDPROM contents are invalid

Initializing  40 megs of memory at addr        0
Boot device: /iommu/sbus/ledma@4,8400010/le@4,8c00000  File and args: /kona
Internal loopback test -- Did not receive expected loopback packet.

Can't open boot device

ok set-defaults
Setting NVRAM parameters to default values.
ok setenv diag-switch? false
diag-switch? =        false
ok 17 0 mkp
ok 08 00 20 c0 ff ee c0ffee mkpl
ok
ok
Target configured. To boot: 'echo reset > /dev/eia2'
:; echo reset > /dev/eia2
:;
```

The device was manually powered on when instructed. Note that for some systems, especially SPARCstations, there is often a lot more output as the system goes through several internal tests, which can take a few minutes. *bootscript will show all that output.*

The most important line above is the one ending with *mkpl*; that sets the MAC and system ID in the Sun's NVRAM. 08 00 20 is common to this generation of Sun systems; for the remainder, use the real address if it's available (often on a sticker printed on the NVRAM chip) or pick something unique to your network.

At the moment, *bootscript* stops short of instructing the Sun to actually boot or reset, but provides the instructions to reboot using the new configuration; this is because you may want to make other changes (like connecting a keyboard to the Sun) before rebooting. *Do not* reboot the system by power cycling it.

5. First Netboot

When the system is reset, it should now attempt to netboot. First, it will perform a *rarp* request, which you should be able to watch `/sys/log/ipboot` to watch `ip/rarpd` respond to.

Having obtained its IP address, the system will attempt to load a kernel using *tftp*. It will ask for a boot file reflecting the IP address and architecture of the system; in the example for *vega*. above, it will ask for `0A011499.SUN4M` To support this without having to maintain a kernel image per system, Plan 9's tftpd offers some extra suppot[2]. tftpd(8) states:

> All requests for files with non-rooted file names are served starting at this directory with the exception of files of the form xxxxxxxx.SUNyy. These are Sparc kernel boot files where xxxxxxxx is the hex IP address of the machine requesting the kernel and yy is an architecture identifier. Tftpd looks up the file in the network database using ipinfo (see ndb(2)) and responds with the boot file specified for that particular machine.

With the example for *vega* given above, the Plan 9 server will respond to a tftpd request for the file *0A011499.SUN4M* from the Sun by sending */sparc/9ss10*.

6. Booting 2nd Edition Plan 9

So far, the procedure described is agnostic to the operating system being served. The original motivation for this was to boot Inferno on the JavaStations and the 2nd Edition of Plan 9 on one of the SPARCstations. We have 2nd Edition Plan 9 booting.

The 2nd Edition of Plan 9 uses an older version of the 9P protocol, which is not served by most of the modern tools. One notable exception is *kfs*, the older on-disk file system.[3]

We have created a kfs file system on a external USB disk, populated that with a 2nd Edition tree, and boot our Sun systems off that. Setting that up is going to be specific to your setup, but The startup script for our Raspberry pi contains a section similar to this:

```
if (grep -s 25A3 /dev/sdU0.0/ctl) {
	disk/partfs /dev/sdU0.0/data
	disk/fdisk -p /dev/sdXX/data > /dev/sdXX/ctl
	disk/prep -p /dev/sdXX/plan9 > /dev/sdXX/ctl
	fossil/fossil -f /dev/sdXX/fossil
	mount -c /srv/fossil.wd /n/wd
	disk/kfs -f /n/wd/tmp/2e.kfs -n 2e && disk/kfscmd -n 2e listen
} &
```

[2] This depends on the bug fix described above.
[3] The other included in the system is *cwfs(4)*, which is generally a better file system, but seemed a bit heavy for this application.

7. Future Work

I would like to be able to netboot some of the systems into operating systems other than Plan 9 or Inferno, in particular OpenStep for the SPARCstations and some flavor of BSD for the Ultra 5 (and possibly one of the JavaStations). The procedure described here should work for the kernel, but more will be needed to get a root file system.