The Various Ports

This document collects comments about the various architectures supported by Plan 9. The system tries to hide most of the differences between machines, so the machines as seen by a Plan 9 user look different from how they are perceived through commercial software. Also, because we are a small group, we couldnt do everything: exploit every optimization, support every model, drive every device. This document records what we *have* done. The first section discusses the compiler/assembler/loader suite for each machine. The second talks about the operating system implemented on each of the various machines.

The Motorola MC68020 compiler

This is the oldest compiler of the bunch. Relative to its competitors commercial compilers for the same machine it generates quite good code. It assumes at least a 68020 architecture some of the addressing modes it generates are not on the 68000 or 68010.

We also use this compiler for the 68040. Except for a few instructions and registers available only from assembly language, the only user-visible difference between these machines is in floating point. Our 68020s all have 68881 or 68882 floating point units attached, so to execute floating point programs we depend on there being appropriate hardware. Unfortunately, the 68040 is not quite so thorough in its implementation of the IEEE 754 standard or in its provision of built-in instructions for the transcendental functions. The latter was easy to get around: we don t use them on the 68020 either, but we do have a library, -l68881 , that you can use if you need the performance (which can be substantial astro runs twice as fast) We don t use this library by default because we want to run the same binaries on both machines and don t want to emulate FCOSH in the operating system.

The problem with IEEE is nastier We didnt really want to deal with gradual underflow and all that, especially since we had half a dozen machines we d need to do it on, so on the 68040 we implement non-trapping underflow as truncation to zero and do nothing about denormalized numbers and not-a-numbers. This means the 68020 and the 68040 are not precisely compatible.

The MIPS compiler

This compiler generates code for the R2000, R3000, and R4000 machines configured to be big-endians. The compiler generates no R4000-specific instructions although the assembler and loader support the new user-mode instructions. There is no support for little-endian machines. Considering its speed, the Plan 9 compiler generates good code, but the commercial MIPS compiler with all the stops pulled out consistently beats it by 20% or so, sometimes more. Since ours compiles about 10 times faster and we spend most of our time compiling anyway, we are content with the tradeoff.

The compiler is solid: we ve used it for several big projects and, of course, all our applications run under it. The behavior of floating-point programs is much like on the 68040: the operating system emulates where necessary to get past non-trapping underflow and overflow, but does not handle gradual underflow or denormalized numbers or not-a-numbers.

The SPARC compiler

The SPARC compiler is also solid and fast. We have seen it do much better than GCC with all the optimizations, but on average it is probably about the same.

We have some old SPARC machines with no multiply or divide instructions, so the compiler does not produce them by default. Instead it calls internal subroutines. A loader flag, -M, causes the instructions to be emitted. The operating system has trap code to emulate them if necessary, but the traps are slower than emulating them in user mode. In a lab where most SPARCS have the instructions, it might be worth enabling the -M flag by default.

The floating point story is the same as on the MIPS.

The Intel i386 compiler

This is really an *X*86 compiler, for *X*>2. It works only if the machine is in 32-bit protected mode. It is solid and generates tolerable code.

Floating point is well-behaved, but the compiler assumes i387-compatible hardware to execute the instructions. With 387 hardware, the system does the full IEEE 754 job, just like the MC68881. By default, the libraries don't use the 387 built-ins for transcendentals; they live in the special -l387 library.

The Intel i960 compiler

This compiler was built as a weekend hack to let us get the Cyclone boards running. It has only been used to run one program — the on-board code in the Cyclone — and is therefore likely to be buggy. There are a number of obvious optimizations to the code that have never been attempted. For example, the compiler does not support pipelining. The code runs in little-endian mode.

The AT&T 3210 Compiler

This compiler generates code for a little-endian AT&T DSP3210. The resulting code has only been executed on peripheral boards, including the commercially available Ariel MP3210 plugged into a PC or Gnot. The generated code is competitive.

The compiler treats double the same as float. By default, it inserts dlock instructions at the beginning of large basic blocks; the -l option to xc turns this off.

The Gnot operating system

The Gnot is a locally-built machine, similar in power and design to a SUN-3, with a two-bit-per-pixel display and no Ethernet. Instead, it uses an AT&T internal twisted-pair network called Incon. The Gnot has no I/O bus, but an external port on the back lets simple I/O boards talk directly to the CPU memory bus. We used this for a handful of applications: local cache disk, printers, etc. Most Gnots have no external peripherals.

We include this port because it would be a good starting point for a port to a SUN-3, provided documentation for the SUN hardware is available. The SUN-3 MMU is related to that of the Sparcstation, so ss/mmu.c might be another good starting component. Sydney University ported an earlier release of Plan 9 to some SUN-3 models, so it might be worth watching their Web page (http://www.cs.su.oz.au/Plan9/) to see if they do it again.

The NeXTstation operating system

Plan 9 runs on the 68040-based NeXTstation but on no other NeXT machine. This is actually one of our favorite ports: the crisp two-bit-per-pixel display is a delight. The one strange part is that we need a three-button mouse, so we unplug the NeXT mouse and use a 9600-baud serial-port Logitech mouse on serial port B.

To get the system running, install NeXT's boot program in the TFTP directory (called /lib/tftpd on Plan 9) and /68020/9nextstation where BOOTP can find it (we put it in /68020/9nextstation). Boot that file from the network by typing ben /68020/9nextstation to the boot ROM. This will load the boot program and use that to load the real operating system. This can be made to happen automatically; see the NeXT manuals for details.

This port could probably be made to run on the old 68030-based cubes without too much work. Like most manufacturers, NeXT kept the architecture pretty similar between models. The 68030 has a very different MMU, though, so the job won't be trivial.

The MIPS Magnum operating system

This is another old favorite: although the tube is nothing like as nice as on the NeXTstation, the MIPS Magnum is all-round a very comfortable place to run Plan 9.

The file to access with BOOTP is /mips/9magnum. Set the console ROM environment variable to m and bootmode to e. Then type bootp()/mips/9magnum to the boot ROM to run the system. To run the machine as a CPU server, rather than a terminal, set console to 1 instead (as a CPU server, the display is not used) and boot /mips/9magnumcpu. There is a disk-resident boot program for this machine; see booting (8) for information.

We have Magnum 3000s and the system will run on only that model. We use 1-bit frame buffers and 8-bit color frame buffers; both run fine from the same binary.

The SGI Power Series operating system

We use SGI Power Series machines as our CPU servers. We don't use them as workstations, so you'll find no code in our system to support SGI frame buffers or the Reality Engines. The machines we have are 4D series with IO3 boards. There is IO2 code in the kernel but it hasn't been tested for years.

The system can be started by using BOOTP to load /mips/9power, but the system behaves better with a two-stage boot; load /mips/9powerboot first. See the manual page *booting* (8) for more information.

The SGI Indigo operating system

The SGI Indigo, with entry-level graphics, is supported as a terminal. Device code is common to the R3000 and R4000 models, but device addresses and memory management are different, so there are separate source directories. Boot the system by using BOOTP to load /mips/9indigo3k or /mips/9indigo4k from a file server. To boot Plan 9 from a local disk, first use *prep* (8) to put a 1MB partition at the beginning of the disk. The files to be placed in this partition are generated by running mk in /sys/src/boot/indigo. To write the partition,

```
cat file >'#w n'/sd nboot"
```

where *file* is either partboot.3k or partboot.4k and *n* is the SCSI id of the disk. To boot from the PROM monitor, type

```
scsi(0)disk(n)b
```

It should be possible for Unix and Plan 9 file systems (with their respective boot partitions) to share a single disk, but we have not attempted this.

The SGI Challenge M operating system

Plan 9 also runs on the SGI Challenge M, but there is no support for the SCSI or GIO buses on the machine. The SGI software uses all the space in the NVRAM so there is no place to write the authentication information to run the system as a CPU server. Therefore, although the system runs fine as a CPU server, it is necessary to type the password, IP address, and so on at the console prompt each time the system is bootstrapped.

To boot the system, use BOOTP to /mips/9chm.

We have not ported the system to the Indigo2 because we don't have documentation for the hardware. However, the Challenge M hardware is closely related to that machine, so this version of the system would be a good starting point for such a port. The main thing necessary would be a driver for the GIO graphics card.

The Sparcstation operating system

We have SUN SLCs and a straight Sparcstation-2; the same binary works on both. It does some auto-configuration at boot time. It makes a couple of assumptions: that there is no external memory and that there is only one frame buffer. The frame buffer it uses is the one with the lowest-numbered SBUS slot; this is also how the ROM selects the main frame buffer. Neither of these assumptions is deep-rooted, but we have very scanty documentation for the machines so we didn't want to try anything risky.

We believe the binary should work on the following machines: Sparcstation 1 (4/60), IPC (4/40), 1+ (4/65), SLC (4/20), Sparcstation 2 (4/75), ELC (4/25), and IPX (4/50), but, as noted, we've only tried two of these. The only frame buffers we know work are the bwtwo and cgsix, but we think the cgthree will work since it is supposed to be the same as the cgsix without acceleration hardware, which we don't support anyway (again, no documentation). Another peculiarity is that, to use the frame buffer, you must set up the ROM's environment variable output-device to be screen so the ROM will enable the video controller; we don't know how.

There is a problem with some very old machines, even of the types we know work: some of the earliest SCSI DMA controllers fail utterly under Plan 9, and others have problems driving some disks, so it might not be possible to use the disks on vintage machines.

We have a similar port for the Sparcstation 10 that should also work on a 20. It has only been tried with a cgsix but should work with a cgthree. Due to the structure of the Plan 9 kernel, we can only have two banks of memory. With less than 64MB in each slot, this limits the system to having memory in two slots. But if the low slot is full, the system can access more memory: the kernel will take the contiguous memory as one bank. I/O space is mapped over SBUS slot 7, so don't put memory in that slot. Note that physical slot number does not equal logical: physical 01234567 correspond to logical 04152637.

SUNs use RARP to translate their Ethernet address to their IP address when preparing to load the kernel into memory. (Most other machines store information in non-volatile RAM and use BOOTP.) They then use TFTP to load a file identified by the hexadecimal IP address and machine architecture. To boot from the network, install /sparc/9ss (/sparc/9ss10 for the Sparcstation-10) as the boot file for the system where TFTP can find it under that name (see *booting*(8)). To boot from a Plan 9 system, just make that the value of the bootf attribute in the network database (see *ndb*(8)). To run the machine as a CPU server, boot /sparc/9sscpu (/sparc/9ss10cpu).

The IBM PC operating system

The PC (and clones) version of Plan 9 can boot either from MS-DOS or directly from a floppy created by the format command; see *prep*(8). Plan 9 runs in 32-bit mode which requires a 386, 486, or Pentium and has an interrupt-driven I/O system, so it does not use the BIOS (except for a small portion of the boot program and floppy boot block). This helps performance but limits the set of I/O devices that it can support without special code.

Plan 9 supports the ISA and EISA buses. There is no direct support for PCI, although PCI versions of some VGA and SCSI cards are known to work. It is infeasible to list all the supported machines, because the PC-clone marketplace is too volatile and there is no guarantee that the machine you buy today will contain the same components as the one you bought yesterday. (For our lab, we buy components and assemble the machines ourselves in an attempt to lessen this effect.) Both IDE and SCSI disks are supported, and there is support for LBA mode on IDE drives. The SCSI adapter must be either an Adaptec 1542[BC] or compatible (e.g., any Buslogic ISA, EISA or PCI controller) or an Ultrastor [13]4f. Supported Ethernet cards include the WD8003, WD8013, SMC Elite, and 3C5[07]9. The NE2000, NE4100, and 3C589 work in some machines. There must be an explicit Plan 9 driver for peripherals; it cannot use DOS or Windows drivers. Also, Plan 9 cannot exploit special hardware-related features that fall outside of the IBM PC model, such as power management, unless architecture-dependent code is added to the kernel. For more details see *plan9.ini*(8).

Plan 9 supports a number of VGA cards using the S3 80[15], S3 928, S3 864, Tseng ET4000, ATI Mach32/64 and Cirrus Logic CLGD-54[23]x chips. For more details see *vgadb*(6) and *vga*(8).

For audio, Plan 9 supports the Sound Blaster 16 and compatibles. CD-ROMs are supported two ways, either on the SCSI bus, or a Mitsumi, Panasonic, or Matsushita CD-ROM drive connected to a Sound Blaster or compatible card. (Note that audio doesn't work under Plan 9 with 8-bit Sound Blasters.)

Plan 9 uses GMT as its time base, while DOS uses local time, so unless you live on the prime meridian you'll need to reset your DOS clock after running Plan 9. Finally, it's important to have a three-button mouse with Plan 9. The system works with either a PS/2 or serial mouse.

Once you have Plan 9 installed (see the separate installation document) run the program b.com from DOS or use a boot floppy. See *booting*(8), *b.com*(8), and *prep*(8) for more information.

If the machine has an MS-DOS Extended Memory Manager installed, b.com tries to uninstall it during the bootstrap, but it's safer to set up the system so the Extended Memory is not installed when b.com starts.

The file server

The file server runs on only a handful of distinct machines. It is a stand-alone program, distantly related to the CPU server code, that runs no user code: all it does is serve files on network connections. It supports only SCSI disks, which can be interleaved for faster throughput. Even on PCs, only SCSI disks may be used, although a DOS file on an IDE drive can hold the configuration information. See *fsconfig*(8) for an explanation of how to configure a file server.

To boot a file server, follow the directions for booting a CPU server using the file name 9*machtype*fs where *machtype* is power, magnum, ss, etc. as appropriate.

The SGI file server

The system runs on 4D multiprocessors with IO3 boards, just like the CPU server system. The system supports the internal LANCE Ethernet controller on the IO3, or you can use an Interphase V/Ethernet 4207 Eagle controller on the VME. It is also possible to use a card made by Cyclone Microsystems to connect VME systems up with a high-speed fiber link. See the file /sys/src/fs/cyc/README for details.

The MIPS 6280 file server

This version of the system is pretty much a port of the SGI system, so has the same properties. We use an Eagle Ethernet controller and an Interphase V/SCSI 4210 Jaguar dual SCSI disk controller on the VME bus. There is no support for the internal SMD drive.

The MIPS Magnum file server

The Magnum file server doesn't use the built-in frame buffer: it requires a 9600 baud terminal connected to tty1. Set the console environment variable to 1 and boot using BOOTP.

The Sparcstation file server

The system assumes the same things about the hardware as the Sparcstation CPU server code. However, the file server doesn't use any existing frame buffer: it requires a 9600 baud terminal connected to ttya. It is known to work only on a Sparcstation 2 (4/75). There is no Sparcstation 10 support.

The problems with old SCSI DMA controllers are critical for the file server. Before committing to a particular Sparcstation as a file server, make sure the disks work under Plan 9.

The IBM PC file server

Except for the restriction to SCSI disks, the PC file server has the same hardware requirements as the regular PC operating system. Any of the supported SCSI and Ethernet controllers may be used, as may any of the boot methods described in *b.com*(8).

To boot any PC, the file b.com must reside on a MS-DOS formatted floppy, IDE disc, or SCSI partition. However, PCs have no non-volatile RAM in which the file server can store its configuration information, so the system stores it in a file on an MS-DOS file system instead. This file, however, cannot live on a SCSI disc, only a floppy or IDE. (This restriction avoids a lot of duplicated interfaces in the system.) Thus the file server cannot be all-SCSI. See *plan9.ini* (8) for details about the *nvr* variable and specifying the console device.

Backup

Our main file server is unlikely to be much like yours. It is an SGI Power Series 4D multiprocessor with 128 megabytes of cache memory, 27 gigabytes of SCSI magnetic disk, and a SONY WDA-610 Writable Disk Auto Changer (WORM), which stores almost 350 gigabytes of data. The WORM is actually the prime storage; the SCSI disk is just a cache to improve performance. Early each morning the system constructs on WORM an image of the entire system as it appears that day. Our backup system is therefore just a file server that lets you look at yesterday's (or last year's) file system.

The SONY WORM is the only one we've tried and therefore the only one for which we have a driver. It shouldn't be too hard to adapt the code to a different brand of jukebox, but there are certainly assumptions about the peculiarities of the device (the things are all weird). Sydney University got another brand of jukebox working under the old system; for information see their URL listed above. You might also consider attaching a

CD-R jukebox or even just using a single WORM drive and managing the dumps a little less automatically. This is just a long way of saying that the system as distributed has no explicit method of backup other than through the WORM jukebox.

Not everyone can invest in such expensive hardware, however. Although it wouldn't be as luxurious, it would be possible to use mkfs (8) to build regular file system archives and use scuzz (8) to stream them to a SCSI 8mm tape drive. Mkext could then extract them.

It is also possible to treat a regular disk, or even a part of a disk, as a fake WORM, which can then be streamed to tape when it fills. This is a bad idea for a production system but a good way to learn about the WORM software. Again, see *fsconfig* (8) for details.