

# INSTALLING BSD OPERATING SYSTEMS ON IBM NETVISTA S40

MICHO ĐURĐEVICH

ABSTRACT. We present several ways of setting up FreeBSD operating system on IBM Netvista S40, a so-called 'legacy free' computer. The difficulty arises because the machine has no standard AT keyboard controller, and the existing FreeBSD subroutines for the gate A20 and for the keyboard controller probes result inappropriate. We discuss a replacement bootstrap code, which more carefully deals with the A20 issue. Some simple modifications to the FreeBSD kernel code are considered, too. A manual method for preparing a bootable installation CD, suitable for both Netvista and all standard configurations, is examined. Installations of DragonFly, NetBSD, OpenBSD and OS/2 are also discussed.

## 1. INTRODUCTION

### 1.1. The Machine



In this note\* we shall talk about installing FreeBSD on a very interesting and elegant machine: IBM Netvista S40. In its creator own terminology, it is 'legacy-free'. The computer has no parallel, serial, AT keyboard, nor PS/2 mouse ports. No floppy controller either. Instead, it has 5 USB ports (2 frontal and 3 rear) connected to a single USB controller. Besides these USB ports, the system only counts with standard video and audio connectors. The video controller is Intel 82810E SVGA and audio chip is Intel ICH 82801AA, both integrated onboard. The CPU is Intel

---

\*Version 5.2005

PIII at 866MHz. The machine is further equipped with a fast Intel Pro PCI network adapter containing a PXE/RIPL boot prom. A quiet 20G Quantum Fireball HDD and a Liteon ATAPI CD-ROM, both connected as masters, constitute the storage subsystem. The case is Flex ATX, a small form factor.

## 1.2. Problem Description

If we try to install FreeBSD 4.11 on this machine in a standard way, by using the bootable installation CD, the computer simply hangs after the initial attempt to read the CD. Nothing is displayed on the screen. Essentially the same symptom appears with NetBSD, DragonFly and FreeBSD 5.3, too. A first natural assumption is that the problem has to do with some BIOS settings. However, no twiddlings in BIOS Setup would remedy this. Another natural path of action is to physically remove the HDD from the Netvista, and connect it to a different ‘auxiliary’ system. This would allow us to install FreeBSD in a standard way. Then put back the HDD in Netvista, and hopefully the machine would boot fine into FreeBSD. Unfortunately, this idea does not work either: the system just hangs as it does during the CD boot attempt. From this we can conclude that, most probably, the bootstrap code is somehow incompatible with the Netvista. In order to verify this, we can use a third approach: Install a boot manager, say GRUB (again, say by using an auxiliary machine) and then try to start the system. At the beginning of the boot process, everything looks encouraging with GRUB. It starts fine and it initializes the FreeBSD loader correctly. Unfortunately, the system freezes as soon as the control is transferred to the kernel. So it follows that the problem lies in both kernel and bootstraps.

In order to deal with such puzzles, we need a mechanism to change efficiently the kernel and the bootstrap files. This is one of the situations where we can take the full and natural advantage of remote booting: Set up a DHCP/PXE remote-boot server, equipped with the full sources, and try to boot the Netvista remotely. It comes as no surprise that PXEboot fails in a similar way—after some rudimentary initial processing the system appears braindead. However, on the remote-boot server, we can easily change the code and quickly deploy all necessary files for the Netvista, in the hope to find some solution of the initialization problem.

## 1.3. Solution

Further experiments and a more detailed examination of the FreeBSD PXEboot source code, show that the subroutine responsible for enabling A20 gate is ‘guilty’. It contains 2 possibly infinite loops, and indeed the Netvista falls into the first of them. The reason is that A20 gate is normally enabled by software, through the keyboard controller. But this machine has no standard keyboard controller, and it falls in a loop of indefinite waiting for the non-existing controller to become ready. The same A20 routine is present in the CD-ROM bootstrap code and also in the hard disk bootstraps (in FreeBSD 5.3, only the hard disk bootstrap code is slightly changed to allow getting out of the deadly loop for systems without a standard keyboard controller). The Netvista does not need any A20 instructions, so removing the subroutine from the code solves the problem of the initialization: the loader would then start fine. In order to be able to boot the kernel, we need to proceed in a similar spirit. Remove references to keyboard and mouse drivers (atkbd & psm), and recompile the kernel. This was the solution for FreeBSD 4.11

and also DragonFly. In the case of FreeBSD 5.3, we are equipped with the flexibility of using device hints. So in order to boot the kernel, we can simply disable atkbd and psm at the loader prompt.

#### 1.4. Organization of the paper

In the next section we shall explain the A20 gate in more detail, and present an alternative A20 subroutine appropriate for both Netvista and the standard (keyboard controller-equipped) systems. In Section 3 we discuss a quick and dirty way to get FreeBSD running on the Netvista, by moving the hard drive on another computer. A more careful approach is to build a bootable FreeBSD installation CD, compatible with the Netvista and also the standard systems. This is the subject of Section 4. In Section 5 some concluding remarks are made. The paper ends with three appendices. In Appendix A we play with DragonFly on the Netvista. The procedure is very similar to installing FreeBSD 4.11. In Appendix B we discuss installing NetBSD, with the help of a live installation CD and GRUB. Finally, in Appendix C we shall leave the framework of BSD, and briefly talk about installing OS/2 and eComStation on the Netvista, by taking advantage of the remote-booting capability of OS/2.

## 2. UNDERSTANDING THE A20 GATE

The problematics of A20 gate is related to the days of ‘prehistory’ of PC hardware. It addresses an inherent incompatibility between 8086 and 80286 architectures. The original IBM 8086 had 20 binary lines for addressing memory (A0-A19). This corresponds to  $2^{20} = 1M$  bytes of addressable memory. The processor 80286 was capable to access physical memory via 24 lines (A0-A23). This corresponds to  $16M$  bytes of addressable memory. In real mode we would only get  $1M + (64K - 16)$  bytes, still above 8086 addressability. This value comes from the expression `FFFF:FFFF` as the maximum address in segmented form, which translates into a linear value of `FFFF0+FFFF=10FFEF=100000+(10000-11)` in hexadecimal form (add one to this number to get the amount of addressable bytes, as we start counting at zero).

By the way, there existed a possibility to run 80286 in protected mode, different from the well known protected mode of 80386—it also is interesting to note that the first versions of OS/2 were running in the protected mode of 286. Processors 80386, 80486 and Pentiums feature a 32-bit address bus, corresponding to the maximum of  $2^{32} = 4G$  bytes of addressable memory.

Returning to our main subject, let us see what would happen if a program designed for 8086 running on 80286 tries to access a memory location requiring more than 20 bits. The 8086 would just ignore the bit 21, corresponding to the line A20 of the address bus. And some old programs required that memory space wraps up at the coordinate  $2^{20}$ . A way for 80286 to effectively simulate this behavior, is to switch off its address line A20. And indeed, IBM introduced such a compatibility function by allowing software to explicitly decide if the address line A20 should be active. Precisely they implemented a logical AND gate on the line A20, where the address line is coupled with an output line of the standard keyboard controller. By setting the keyboard controller line output to 1/0, we effectively enable/disable the gate. By default, the PC would initialize with A20 disabled. However, in order for the CPU to access the full memory space, it is obvious that in this case the A20 gate

must be enabled by the operating system. Otherwise, weird things would happen: All the theoretical memory space would be factorized over the equivalence relation, obtained by flipping bits at position 20. Any reasonably designed operating system would most probably crash under such conditions...

On the other hand, the Netvista S40 is a 'legacy free' system, and it has no such a special gate for A20. It actually lacks any standard keyboard controller (both keyboard and mice are USB). Execution of a straightforward A20 enablement routine on such systems could easily result in weird things. Indeed, this is what happens with the existing FreeBSD and DragonFly code.

Here comes the listing of an alternative A20 enablement subroutine, that should work both for standard systems and the Netvista S40. This code should be put in place of the existing A20 subroutine in files

```

/sys/boot/i386/cdboot/cdboot.s    /sys/boot/i386/pxeldr/pxeldr.s

set_a20:                          cli
                                  movw $qfbsd, %si    # Say something.
                                  callw putstr
                                  movw $0x10, %cx     # Initial wait, to allow
                                  callw delay         # a keypress.
                                  movb $0x1, %ah      # If a key has been
                                  int $0x16           # pressed then skip.
                                  jnz skip_a20
                                  movw $qstar, %di    # Entering the first wait
                                  callw wait_kbdc    # cycle, for the controller
                                  jcz  skip_a20      # to report its readiness.
                                  movb $0xD1, %al     # We are ready to ask for
                                  outb %al, $0x64    # a write command...
                                  movw $qnumb, %di   # Wait for the controller
                                  callw wait_kbdc    # to get ready, again.
                                  jcz  skip_a20
                                  movb $0xDF, %al    # Everything looks ok, let us
                                  outb %al, $0x60    # turn on the kbd line of the
                                                  # gate.

skip_a20:                          movw $monkey, %si
                                  callw putstr
                                  sti
                                  retw

wait_kbdc:                          movw $0x10, %cx

wait_kbdc.2:                        inb $0x64, %al

```

```

        testb $0x2, %al
        jz wait_kbdc.3

        movw %di, %si
        callw putstr

        pushw %cx
        movw $0x2, %cx
        callw delay
        popw %cx

        loop wait_kbdc.2
wait_kbdc.3:    retw

delay:        movb $0x86, %ah      # BIOS wait call. The cx:dx
              int $0x15          # holds the number of
              retw              # microseconds to wait.

...
qfbsd:        .asciz "\r\nA20 Gate [Keypress to Skip]\r\n"
qstar:        .asciz "*"
qnumb:        .asciz "#"
monkey:       .asciz "@\r\n\n"

```

Various of the above lines are optional. They introduce some rudimentary verbose output, so we know whenever the controller is not ready to take command. It also allows to skip the A20 related code altogether, by pressing a key when an initialization message appears. This might be useful to avoid the mess, for example if we plan to expand the A20 code, by introducing different methods of enabling A20 gate, or in highly non-standard configurations where the in/out ports are different.

### 3. MOVING HARD DRIVE BACK AND FORTH

A direct, although non-elllegant way of installing FreeBSD is to move the hard disk on a different machine which is able to boot fine the installation CD. Once we have the operating system up and running on this auxiliary machine, we execute Netvista specific adjustments and return back the hard disk :)

Here is one working scenario, assuming we install FreeBSD 4.11 and the auxiliary machine holds only one (the Netvista) hard drive during the installation.

**Step One:** Install, as usual, the operating system and full sources on the auxiliary machine.

**Step Two:** Patch the assembler file

```
/usr/src/sys/boot/i386/boot2/boot1.S
```

and completely remove the existing A20 routine from it. From the directory /usr/src/sys/boot/i386/boot2 execute

```
make install
```

This would build the stage-1 and stage-2 bootstraps, the files /boot/boot1 and /boot/boot2 respectively.

**Step Three:** Update the bootstrap code on the disk, by copying the bootstraps to the appropriate init areas of the disk slice.

```
disklabel -B ad0s1
```

**Step Four:** Compile and install a new kernel, suitable for the Netvista. The essential thing here is to comment out the following two lines from the kernel configuration file:

```
# device atkbd0 at atkbd? irq 1 flags 0x1
# device psm0 at atkbd? irq 12
```

It is very important not to reboot after step 3, as FreeBSD would most probably crash on an auxiliary machine (which would most probably have A20 disabled at boot time). After step 4, the disk should be ready to boot FreeBSD on the Netvista (but unusable on the auxiliary machine)... A more sophisticated approach is presented in the next section.

#### 4. CREATING A COMPATIBLE INSTALLATION CD

Our objective here is to create a FreeBSD installation CD that could be used on both Netvista and other systems. This should be done on a separate computer, with full sources installed. There exists a full infrastructure for creating custom FreeBSD releases (see `release(7)` and `/usr/src/release`). However in what follows we shall use a simple ‘manual’ method. We shall distinguish 3 slightly different scenarios: FreeBSD 4.11, FreeBSD 5.3 and DragonFlyBSD. In all three cases, we need to create a new installation disk, by replacing the file `/boot/cdboot` with the custom-prepared counterpart. We assume that the bootstrap files have been compiled, in accordance with the previous instructions (new A20 for the cd/pxe bootstrap and complete A20 code removal for HDD stage-1/2 bootstraps).

**Step One:** Copy the contents of the entire CD to a separate directory. For example (after inserting the standard install CD)

```
mount /cdrom; mkdir /work/qvista
cpdup -vvv /cdrom /work/qvista
```

**Step Two:** Place the bootstrap files in the CD directory. For FreeBSD 4.11:

```
cd /usr/obj/usr/src/sys/boot/i386
cp boot2/boot1 /work/qvista/boot/boot1.ns40
cp boot2/boot2 /work/qvista/boot/boot2.ns40
cp cdboot/cdboot /work/qvista/boot
```

In addition, we should copy our custom-prepared kernel to the CD root:

```
cd /usr/obj/usr/src/sys/compile/qvista
cp kernel /work/qvista/kernel.ns40
```

As mentioned in the previous section, we should remove the atkbd/psm devices from the kernel,

```
# device atkbd0 at atkbd? irq 1 flags 0x1
# device psm0 at atkbd? irq 12
```

in order for it to boot properly. In addition, this kernel should contain

```
options MD_ROOT
options MFS
pseudo-device md
```

If we deal with FreeBSD 5.3, then we only have to copy the `cdboot` file. And optionally, we can also

```
cd /usr/obj/usr/src/sys/boot/i386
cp boot2/boot /work/qvista/boot/boot.ns40
```

**Step Three:** Create the ISO-image of the new installation CD:

```
mkisofs -R -no-emul-boot -b boot/cdboot -c boot/boot.catalog
-o /work/diskimgs/qvista.iso /work/qvista
```

**Step Four:** Burn the ISO-image. Assuming that we have an ATAPI CD-burner attached at `/dev/acd0`, then

```
burncd -f /dev/acd0 -s max -v data /work/diskimgs/qvista.iso fixate
```

With this new CD, the Netvista should boot without problems. In the case of FreeBSD 5.3, we would need to escape at the loader prompt and define the following hints:

```
set hint.atkbd.0.disabled="1"
set hint.psm.0.disabled="1"
```

After the OS installation is completed, we should define the above settings in `/boot/device.hints` file. Alternatively, when compiling the custom kernel for the Netvista, we can then remove the options 'atkbd' and 'psm' from the kernel configuration file.

When installing FreeBSD 4.11 on Netvista S40 using the prepared CD, we should escape to the loader prompt, then

```
unload
load kernel.ns40
load -t mfs_root /boot/mfsroot
boot
```

The installation should work more or less as usual. However, after the installation of the operating system completes, we have to do one more step. Move to the emergency holographic shell (Alt-F4) and execute:

```
cp /dist/kernel.ns40 /kernel
cd /dist/boot
/sbin/disklabel -B -b boot1.ns40 -s boot2.ns40 ad0s1
```

In the case of FreeBSD 5.3, there is only one final and optional step:

```
/sbin/disklabel -B -b boot.ns40 ad0s1
```

However, the existing bootstrap code of FreeBSD 5.3 already contains a cx-based timeout loop in its A20 routine. The above step would simply spare your Netvista from executing a bunch of useless cycles every time it boots :)

## 5. CONCLUDING REMARKS

We have had to overcome 2 obstacles, both related to the absence of the standard keyboard controller. The A20 obstacle, related to deadly loops at the very beginning of the boot process, and the kernel obstacle, related to the i8042 keyboard controller code for the FreeBSD kernel. Two kernel subroutines are guilty: `atkbd_attach` and `atkbd_configure`. We shall now present additional simple solutions to deal with FreeBSD 4.11/DragonFly, that do not require including a custom-prepared kernel for the Netvista on the CD. The main idea is to modify the kernel code, so that defining a new loader variable, say `at.kbdc.disabled`, would control whether or not the deadly subroutines get executed.

Here are lines of the code to include in the file `/sys/isa/atkbd_isa.c`:

```
static int kbdc_disabled = 0;
TUNABLE_INT(at.kbdc.disabled, &kbdc_disabled);

...
static int
atkbd_attach(device_t dev)
{
...
    if (kbdc_disabled == 1)
        return 0;
...
}
```

This allows to effectively control the execution of the routine `atkbd_attach`, by setting `at.kbdc.disabled=1` at the loader prompt. However, the above code does not work for `atkbd_configure` (defined in `/sys/dev/kbd/atkbd.c`) as this function gets executed at a too early stage of the kernel initialization (in order to provide keyboard support for visual userconfig module). Instead of using the `TUNABLE_INT` macro, we should access directly the kernel environment variables (inherited from the loader, via the `bootinfo` structure).

```
static int
atkbd_configure(int flags)
{
...
    if (getenv("at.kbdc.disabled") != NULL)
        return 0;
...
}
```

Another possibility is to take advantage of kernel's user configuration files. No changes are necessary to `atkbd_isa.c`, however the function `atkbd_configure` still has to be patched, say as above. Just prepare the install CD, along the lines of the previous section, but remove `userconfig.script_load="YES"` from `/boot/loader.conf`. Together with the kernel, include `/boot/netvista.conf`:

```
di atkbd0
di atkbd0
di psm0
```

At the loader prompt, we would then execute

```
set at.kbdc.disabled=1
load -t userconfig_script /boot/netvista.conf
```

The above mentioned keyboard controller backdoor routine executes too early, so the kernel userconfig script alone would not prevent the disaster. This is why we still need to set the variable `at.kbdc.disabled`. An alternative is just to get rid of this early probing code, assuming we do not need interactive userconfig on any of the systems we would be dealing with.

At the end of the installation process we would then only have to update the bootstraps, as explained in the previous section. Actually, using the holographic shell, we can compile and install the appropriate Netvista kernel even before the first reboot.

It is interesting to observe that the generic kernel of NetBSD does not exhibit problems on the Netvista. In contrary, it detected correctly all the components of the hardware. The only real obstacle is A20-related (the standard installation CD does not boot at all). A procedure for installing NetBSD is presented in Appendix B.

And what about OpenBSD? Well, the answer is pretty simple: To my very pleasant surprise, OpenBSD 3.6 installed just fine, out-of-the box. As with NetBSD, all hardware components of the Netvista were correctly identified by the OpenBSD generic kernel.

## APPENDIX A. DRAGONFLY INSTALLATION

The procedure to prepare the bootable installation CD is virtually identical to FreeBSD 4.11. To boot the Netvista kernel off CD, exit to the loader prompt and then:

```
unload
boot kernel.ns40
```

For the sake of completeness, let us overview here a manual installation of DragonFly, which will use the whole hard disk space. More detailed explanations on installing DragonFly can be found in the DragonFly Hangbook. We first initialize the MBR for such a dedicated disk, and create the BSD slice with the appropriate disk label. As in the case of FreeBSD, we here use the custom-prepared bootstraps, without any A20 routine.

```
fdisk -I ad0
disklabel -w -B -b /boot/boot1.ns40 -s /boot/boot2.ns40 ad0s1 auto
disklabel -e ad0s1
```

Here is a sample partition section of the label data:

	size	offset	fstype	[fsize	bsize	bsp/cpg]
a:	256m	0	4.2BSD	1024	8192	99
b:	512m	*	swap			
c:	<should be left as-it-is>					
d:	256m	*	4.2BSD	1024	8192	99
e:	4096m	*	4.2BSD	2048	16384	99
f:	*	*	4.2BSD	2048	16384	99

Next, we format the created partitions (-U turns the softupdates on):

```
newfs ad0s1a
newfs -U ad0s1d
newfs -U ad0s1e
newfs -U ad0s1f
```

Let us create the mount points and copy the relevant files from the installation CD:

```
mount ad0s1a /mnt
mkdir /mnt/var /mnt/usr /mnt/home
```

```
mount ad0s1d /mnt/var
mount ad0s1e /mnt/usr
```

```
cpdup -vvv / /mnt
cpdup -vvv /etc /mnt/etc
cpdup -vvv /dev /mnt/dev
```

```
cpdup -vvv /var /mnt/var
cpdup -vvv /usr /mnt/usr
```

The optional `-vvv` switch is used to display all objects as they are copied. Furthermore, we have to create the appropriate `fstab` file in `/mnt/etc`. The one associated to the above mentioned disklabel looks like:

# Device	Mountpoint	Fstype	Options	Dump	Pass#
/dev/ad0s1b	none	swap	sw	0	0
/dev/ad0s1a	/	ufs	rw	1	1
/dev/ad0s1d	/var	ufs	rw	2	2
/dev/ad0s1e	/usr	ufs	rw	2	2
/dev/ad0s1f	/home	ufs	rw	2	2

Finally, let us adjust our kernel so that the Netvista will boot properly by default.

```
cd /mnt
mv kernel kernel.ORIGINAL
mv kernel.ns40 kernel
rm boot/loader.conf
```

At this point the Netvista should be ready to happily boot the fresh DragonFly installation.

## APPENDIX B. NETBSD INSTALLATION

We present here a method based on creating a live NetBSD install CD, and using GRUB as the bootloader. A nice starting point is the live CD by Joerg Braun (it can be downloaded from the NetBSD distribution sites). The CD should be a complete working text-mode NetBSD system, containing all NetBSD distribution sets, a complete GRUB install, and the full kernel sources. I used the following command to prepare the ISO-image:

```
mkisofs -U -R -iso-level 3 -max-iso-filenames -no-emul-boot
  -boot-load-size 30 -boot-info-table -c boot/boot.catalog
  -b boot/grub/stage2_eltorito -o $diskimage $cdsource
```

I recommend to configure the CD for bash as the default shell, and with at least 2 virtual terminals enabled. From now on it is assumed that we have successfully booted such a CD-based NetBSD system.

```
fdisk -u /dev/wd0
disklabel -e -I /dev/wd0
```

Here is the relevant section of the initial disklabel (compare it with DragonFly labels):

```
7 partitions:
#   size   offset  fstype      [fsize   bsize   bsp/cpg]
a:  524288     63   4.2BSD      1024     8192     512
b:  524288  524351   swap
c: 39876417     63  unused      0         0
d: 39876480     0  unused      0         0
e:  524288 1048639   4.2BSD     2048    16384     512
f:  8388608 1572927   4.2BSD     2048    16384     512
g: 29914945 9961535   4.2BSD     2048    16384     512
```

We can play within another virtual terminal to verify calculations involving large quantities of sectors:

```
#echo $((256*1024*2))
524288
#echo $((63+2*524288))
1048639
#echo $((63+3*524288))
1572927
#echo $((4096*1024*2))
8388608
#echo $((8388608+1572927))
9961535
#echo $((39876417-8388608-3*524288))
29914945
```

The following instructions build the filesystems:

```
newfs /dev/wd0a
newfs -O2 /dev/wd0e
newfs -O2 /dev/wd0f
newfs -O2 /dev/wd0f
```

Now, we have to create mount points, and bring in the newly created partitions:

```
mount /dev/wd0a /mnt
mkdir /mnt/proc /mnt/kern /mnt/usr /mnt/var /mnt/home
mount /dev/wd0e /mnt/var
mount /dev/wd0f /mnt/usr
```

We are ready to copy the software on the harddisk:

```
cd /i386/binary/sets
tar xvzpf $distname -C /mnt
```

where `$distname` is the appropriate distribution set. We have at least to include the base, etc and kernel sources packs (or just a kernel binary, assuming we prepared the kernel with root filesystem info compiled-in). As our next step, let us create all device nodes.

```
cd /mnt/dev
./MAKEDEV all
```

The boot loader should be installed and configured for the appropriate kernel.

```
cp -r /boot/grub /mnt/boot
vi /mnt/boot/grub/menu.lst
grub
grub> root (hd0,0,a)
grub> setup (hd0)
grub> quit
```

Now it comes the only really Netvista-specific step. We have to compile the appropriate kernel, including the root filesystem information. The compilation step is necessary because the NetBSD kernel exhibits troubles to take the input from the USB keyboard (it appears locked in a loop) if we would have to specify the root filesystem and init data at boot time.

```
chroot /mnt /bin/csh
cd /usr/src/sys/arch/i386/conf
cp GENERIC QVISTA
vi QVISTA
```

The following configuration line defines the root filesystem to the kernel:

```
config          netbsd    root on wd0a type ffs
```

What follows are the standard kernel compilation steps:

```
config QVISTA
cd /usr/src/sys/arch/i386/compile/QVISTA
make depend
make
cp netbsd /
```

The system is almost ready for the first hard-disk boot. We have to create the `/etc/fstab` file:

# Device	Mountpoint	Fstype	Options	Dump	Pass#
kernfs	/kern	kernfs	rw		
procfs	/proc	procfs	rw		
/dev/wd0b	none	swap	sw	0	0
/dev/wd0a	/	ffs	rw	1	1
/dev/wd0e	/var	ffs	rw,softdep	2	2
/dev/wd0f	/usr	ffs	rw,softdep	2	2
/dev/wd0g	/home	ffs	rw,softdep	2	2

Finally, we must define `rc_configured="YES"` in `/etc/rc.conf`. And optionally, adjust a couple of other basic configuration parameters.

## APPENDIX C. OS/2 INSTALLATION

I recently tried to install OS/2 MCP2 on my Netvista S40. The installation CD booted fine, but very quickly the install program got pretty disturbed by the existing BSD hard disk layout, and the whole experiment terminated by a trap :(

Historically, my first attempt to install something non-trivial on this faithful machine was during Q-Systems participation in the eComStation project. I was interested to install Version 1.0 of eComStation. I started the installation program from the eComStation CD, and selected USB option on the preboot screen. Unfortunately, the USB drivers included in this CD were not compatible with the USB controller on the Netvista. The system booted fine (modulo some USB error messages) but both keyboard and mouse were unusable. I had to find better USB drivers. But how to boot the system with these better drivers? I had no any USB floppy drive with me.

A natural solution was to DHCP/PXE boot the Netvista from another eComStation system, making sure it would load all appropriate drivers, and then launch the installer program, and install the operating system locally.

This is the basic idea of a network install. A great advantage of such an installation method is that we have full control over what and how is really loaded on the target system. And once it works fine, the install can be easily deployed on multiple identical/similar configurations, over complex networks. A detailed discussion of remote-booting eComStation from eComStation can be found in our OS/2 PXE remote-boot article. Here we shall touch only Netvista-specific things.

At first, we have to enable USB support (at least for keyboard and mouse). This is done by including the following lines in the `CONFIG.SYS` file:

```
BASEDEV=USBD.SYS /I13
BASEDEV=USBHCD.SYS
BASEDEV=USBHID.SYS
```

```
DEVICE=C:\OS2\BOOT\USBKBD.SYS
DEVICE=C:\OS2\BOOT\USEMOUSE.SYS
```

Now, there is a subtle point regarding USB drivers. I used MCP2 USB drivers \*except\* `USBHCD.SYS`, which I took from the eComStation install CD. Only in this way the system booted properly. The `USEMOUSE.SYS` driver must be loaded after `MOUSE.SYS`.

The HDD and CD-ROM related entries are:

```
BASEDEV=OS2DASD.DMD
BASEDEV=OS2LVM.DMD
BASEDEV=CHKDSK.SYS
BASEDEV=DANIS506.ADD
BASEDEV=DANIATAP.FLT
```

```
DEVICE=C:\OS2\BOOT\OS2CDROM.DMD /Q
IFS=C:\OS2\BOOT\HPFS.IFS
IFS=C:\OS2\BOOT\CDFS.IFS
```

There is a strange correlation between (non-existent) floppy drive and LVM program. If the following line

```
BASEDEV=IBM1FLPY.ADD /A:0 /FORCE:1 /U:0 /F:1.44MB
```

is absent, then the system would *completely freeze* when exiting LVM, assuming that a volume was created during the LVM session.

In order to launch the installation program properly, it is necessary to set the following variables:

```
SET RUNNINGECSINSTALL=YES
SET CD_DRIVELETTER=S
```

The following variables alter the behavior of the installation program, regarding the creation of the appropriate `CONFIG.SYS` file for the local installation. If we do not define these variables, nothing spectacular would happen. The installation program would simply fail to introduce a bunch of critical entries in the client `CONFIG.SYS` file, and we would have to introduce them manually.

```
SET CSM_DOSSUPPORT=TRUE
SET CSM_WARPENTER=TRUE
SET CSM_OS2DASD=TRUE
SET CSM_ATAPI=1
SET JJSCDROM=FALSE
SET CSM_IDEDRIVER=1
```

The installation program is then started by invoking `eCSMakeDisk`. The first phase of the installation process passed smoothly. Before rebooting the system for the first time (the install process continues from the corresponding local volume), it was necessary to make several critical adjustments. The rest of the installation process went smoothly, in a straightforward way.

### C.1. `CONFIG.SYS`+USB Files

At first, I had to enable the corresponding USB entries in the Netvista local `CONFIG.SYS` file. Finally, I had to copy the correct USB driver files (see above), from the eComStation server to the `OS2/BOOT` folder of the installation volume.

### C.2. Replacing `OS2LDR`

Surprisingly, it turned out that the loader file installed locally fails to boot. The symptom is that when we try to boot Netvista from the corresponding installation volume (after completing the first phase of the install process) nothing happens, we get just a blinking cursor. A more detailed analysis reveals that the cause for this is exactly the same as for the troubles with FreeBSD: An ugly A20 enablement code, which simply hangs due to the absence of the standard keyboard controller on the Netvista S40.

The solution is to replace original `OS2LDR` file (found in the root of the installation volume) with the magical `OS2LDR` file from FP.26. This loader file is much better designed, simpler, and it is also indispensable to use it when booting eComStation from eComStation, as explained in our `OS/2 remote-boot` article.

### Acknowledgements:

I would like to express my special gratitude to a great Mexican pilot, my friend Antonio Vargas.

INSTITUTO DE MATEMATICAS, UNAM, AREA DE LA INVESTIGACION CIENTIFICA, CIRCUITO EXTERIOR, CIUDAD UNIVERSITARIA, MÉXICO DF, CP 04510, MEXICO

*E-mail address:* `micho@matem.unam.mx`

`http://www.matem.unam.mx/~micho`