Porting AIX onto the Student Electronic Notebook

John Ioannidis, Gerald Q. Maguire Jr.
Israel Ben-Shaul, Marios Levedopoulos, Micky Liu

Department of Computer Science
Columbia University

Technical Report No. CUCS-042-90

December 12, 1990

Abstract

We describe the Student Electronic Notebook and the process of porting IBM's AIX 1.1 to run on it. We believe that portable workstation-class machines connected by wireless networks and dependent on a computational and informational infrastructure raise a number of important issues in operating systems and distributed computation (e.g., the partitioning of tasks between workstations and infrastructure), and therefore the development of such machines and their software is important. We conclude by summarizing our activites, itemizing the lessons we learned and identifying the key criteria for the design of the successor machines.

1 Introduction

The **Student Electronic Notebook** project is a joint effort between IBM and Columbia University. The reason we called the project, and also the machines that we built, 'Student Electronic Notebook' is that we intend such machines to replace the traditional student notebooks. The plan was (and is) that students be able to go through their academic careers only needing such a decice, rather than a collection of paper notebooks, texbooks, class notes, handouts, and terminals. Students should be able to carry one with them at all times, in class, in the lounge, around campus, in their dorm room, etc., and do all their reading, notes-taking, and even homework, on them. While this was initially targeted at students, we have come to realize that there is a much larger potential user base for portable, wireless workstations. We describe our visions for what we call the *Personal Information Portal* in another paper [IM90b].

In this paper we concentrate on the software and hardware effort of porting an existing operating system (IBM's AIX 1.1) onto our prototype hardware. We believe that our experience will be of use and give insights to anyone trying to design a portable workstation and an operating system for it.

Our boundary conditions for a prototype were:

- The unit must be roughly the size and weight of a standard student looseleaf binder.
- The unit should be portable.
- It should be built using existing components.
- It should have no hard disk in order to conserver power and weight.
- The primary input device should be a paper-like interface.
- A keyboard should be optional
- The machine should be sufficiently powerful to run an operating system. That is, we wanted it to be more than a graphics terminal or a windowing terminal (e.g., an "X-terminal").
- The network throughput should be as high as possible.

The resulting prototype is described in the following section. It is not a product and was not designed to be a product, but rather it is a rough prototype. The audio analogy is that it is more like the first Edison phonograph than a modern, high-end audio system.

Given the hardware, we were faced with the problem of porting an operating system to run on it. Several selection criteria applied here:

- We wanted to provide a multitasking environment.
- Since we did not know what the partition of tasks would be between the infrastructure and the portable unit, we wanted an operating system.
- We did not want to develop an OS from scratch, as this was not our focus.
- We wanted an OS that would run on our hardware, for which we could get source code and with which we had experience, and that also had the ability to run MS-DOS programs as user processes. The best fit for our needs was IBM's UNIX offering, also known as AIX.

It should be noted that, while these criteria were self-evident to us, we had to argue considerably to convince the non-technical people to allow us not to use DOS.

It turned out that our expectations of the software were not unreasonable; to wit, as of July, 1990, the SEN can boot over the radio interface, run AIX diskless, and run X11R4, using a stylus instead of (or in addition to) a mouse, a pop-up keyboard window and/or a keyboard. The same software will also run on a stock PS/2-80 or PS/2-55sx; in fact, we routinely utilize such machines to gateway network traffic between the SEN radio network and our departmental ethernet, and through that, to the rest of the internet. We also use those PS/2-80s as our infrastructure machines, and as our development platform for new software.

2 The Hardware

The prototype SEN unit measures 10 x 11.5 inches and weighs 10 pounds. It uses the motherboard from an IBM PS/2 model 55sx with eight megabytes of battery-backed memory, a 640x480x4 LCD display connected to a Yamaha EGA/VGA-compatible card, a Microtouch resistive overlay screen with a conductive stylus, and a Telesystems ARLAN-450 radio network interface card. Finally, the unit can be powered either using internal

or external NiCd rechargeable batteries, or it can be connected to a DC power supply. It can run on batteries for about an hour and, when turned off, the unit can retain the contents memory for more than two days.

Let us now describe in more detail the hardware platform.

The motherboard has a couple of modifications. First, its standard memory (all 8 Megabytes) is battery-backed. There is a special circuit connected to the memory modules that will periodically refresh the memory when the SEN is turned off. The SEN also has a modified BIOS. The two major differences are:

- The ability to use the parallel port as a floppy disk. To that end, a special DOS program, called IPLNOTE, is needed to run on a 'docking station', typically a standard PS/2-55sx. The parallel port of the SEN is connected to the parallel port of the docking station, and the latter's floppy disk drive appears as the SEN's floppy disk drive.
- On power-on, if a special flag is set, the BIOS will skip the power-on self-tests (one of which clears memory) and jump to a predetermined place in memory, thus allowing for "warm restarts".

The display of the unit is a 640x480 16-level grayscale LCD screen connected to a Yamaha VGA/EGA-compatible display adapter. The BIOS of this card is responsible for disabling the on-board VGA controller of the motherboard so that the system can use it (the Yamaha) as its master display.

On top of the display is mounted a Microtouch resistive overlay screen. Its controller is inside the SEN and is connected to the serial port. The controller can give pen up/down information, as well as x-y coordinates, with a resolution of 1024x1024. It can operate in polled mode, or can supply a continuous stream of x-y coordinates.

The radio interface is a full-size PC card that provides a nominal bandwidth of 230 kilobits per second and operates in the 902 MHz to 928 MHz band using Spread-Spectrum technology. The ARLAN cards provided the highest bandwidth we could obtain from off-the-shelf components without FCC licencing. Their range is over 150 feet in an office environment, and up to six miles with a directional Yagi antennas and line-of-sight view. These cards are obviously too big to fit inside the SEN enclosure, and they are housed in a separate enclosure connected to the back of the unit.

In addition to all this non-standard hardware, there were two more problems with the PS/2-55sx motherboard that affected our porting effort. The machine has no "2 kilobyte configuration NVRAM" from which the boot program and the AIX initialization code takes its configuration information, nor is the PS/2-55sx hard disk supported under AIX 1.1. The latter did not affect the SEN software (since the machine is diskless), but it did delay the porting effort since we could not utilize the local hard disk during porting to the PS/2-55sx.

3 System Software

As we described in the introduction, the system software consists of the booting code, the patches necessary for diskless operation, the radio interface, the display and the stylus support. We shall now examine these components in detail.

3.1 Diskless Booting

Booting of diskless workstations is a well-understood process [CG85], [SMIb]. It consists of two stages; the primary loader, which resides in firmware and is given control on power-up, and which determines the boot device and proceeds to load the secondary loader from there. The secondary loader is responsible for bringing in the kernel image, setting up the virtual memory environment and transfering control to the operating system.

In our case, we wanted more functionality in the first half of the boot process than the firmware provided, but we did not have the ability to change it, so we split the "first phase" in two, resulting in a three-stage boot:

- The firmware either loads and runs the primary boot code, or jumps directly to the secondary boot code, if the latter is already in memory.
- If not, the primary boot code loads the secondary boot code.
- The secondary boot code loads the kernel image and the RAM-disk-based filesystem (more about that later), sets up the virtual memory environemt, and gives the kernel control.

When a SEN is first powered up, the modified BIOS checks whether the SEN is connected to a docking station. If so, it will proceed with a cold boot from the docking station's floppy. Otherwise, it will check whether the warm-start flag is enabled; if so, it will jump to a predefined location in low memory (remember that the SEN has battery-backed RAM), otherwise it will fall through to BASIC.

The primary boot code, called `boot0`, resides in the boot sector (sector 0) of the floppy. It is limited in size to 512 bytes, and can therefore only do simple things. It can interpret the contents of the floppy as a DOS floppy or as a UNIX file system. In the latter case it will read in the file called `/boot` and give it control.

`/boot` is the secondary loader. It is a specially-linked `.EXE` (DOS) program which, however, does not depend on `DOS` being present. If it was loaded by the primary boot code, it will set the warm-boot flag and establish pointers in system memory so that it will be entered directly if there is another power-up cycle.

`/boot` lets us to do a number of things:

- Load and run
  - DOS.
  - a (specially-linked) DOS .EXE file.
  - a UNIX COFF (again, specially linked) file,
- optionally, load a data file into the BSS segment of the loaded COFF file.

The above files can be loaded using either the docking station's floppy or the radio network interface. Loading via radio is only slightly faster than loading from floppy, but it has two big advantages:

- A SEN does not need to be physically at a docking station to boot
- Using specially-developed broadcast protocols [IM90a], any number of SENs can boot simultaneously, via the radio, in roughly the same time it takes for one to boot.

We went through three distinct phases while developing the booting code.

Initially, we would load a kernel with a RAMdisk precompiled in it. This limited us to a very small RAMdisk (about 400K), because the entire kernel had to fit on a single floppy. We used that as our initial approach as it did not require a modified primary loader.

This was not adequate for use with a larger RAMdisk, so we modified the secondary loader to locate a particular address in the kernel's `.bss` segment, namely, the address of the start of the RAMdiskdata, and load the contents of a second file (from a separate floppy) starting at this offset. Since the `.bss` space is allocated by consulting a header entry (rather than reading its contents from the COFF file itself), this enabled us to have a RAMdisk large enough to contain all the programs needed to eventually boot AIX diskless.

Although this approach worked successfully, in fact we used this method for about three months, it was clearly not adequate in that we still had to depend on a parallel connection to a host machine in order to boot, and booting from a floppy was painfully slow (approximately six minutes).

The booting method we intended to use from the very beginning, but because of various problems did not get it running until after the rest of the system was in place, was this: Using the primary loader (the boot block), we load a .EXE program (developed under DOS) that can access the radio card. It, in turn, queries a simplified BOOTP-like [CG85] protocol, acquires the internet address of the SEN and then transfers the kernel and the RAMdisk contents using CFDP which is another locally developed protocol [IM90a].

CFDP (short for "Coherent File Distribution Protocol) is a protocol that takes advantage of the fact that our traffic is broadcast, and that, in the perceived mode of operation, many SENs may be trying to boot at the same time. A booting SENfirst checks whether there is already boot-related traffic in the network. If so, it grabs the boot packets (which are being broadcast by the boot server) and stores them in memory. When it stops receiving packets, it checks its tables to see what packets are missing, and then requests them in a request vector. The big advantage of this method is that booting n SENs does not require time proportional to n, but rather at most twice the time it takes to boot one.

The first time the secondary loader is read in memory, it adjust some firmware parameters to hide itself from the kernel initalization code, as if it were part of the firmware-reserved memory. It also sets the warm-boot vector to contain a JMP instruction to itself, so that power-cycling the machine gives control to the secondary loader.

The next step would be to write ROMable code for both loaders and incorporate it in the boot EPROMs. We thought that this is more of an product issue rather than a research issue, so we didn't pursue it further.

Finally, the goal is to modify the kernel to be able to checkpoint itself, and modify the secondary boot program so that it knows about this "freeze" function, and how to hot-start a suspended kernel without having to reboot it.

3.2 Diskless operation

One of our objectives was totally diskless operation. The arguments for and against diskless workstations are many and varied, but our chief reason was portability and power consumption; disk drives are still heavy, bulky and they consume a lot of power (which also increases the size of the power supply or batteries, adding to the overall weight of the machine).

3.2.1 Root File System

Unlike other modern UNIX systems (SunOS, HP-UX, etc.), AIX 1.1 had no support for diskless operation. It does support NFS, but the root filesystem cannot be NFS-mounted; it has to be a block device. We have already explained how to boot a kernel without the need of a permanently attached local disk. Naturally, if we were going to have completely diskless operation, we needed some way of having a root filesystem that was not on any kind of "hard" device. We considered the following options:

1. Write a device driver that looks like a disk drive to the file system, but forwards requests for blocks through the network interface to a *'network disk'* server. This is the original SunOS approach, implemented by what was known as the **nd** protocol [SMIa].

2. Have a minimal, memory-resident 'RAMdisk' file system that contains the absolute minimum scripts needed to startup the kernel, then NFS-mount all the necessary file systems from an ordinary NFS server.

3. NFS-mount the root file system from the beginning; (e.g., SunOS 4.0 uses this method).

4. Allow for remote device access (à-la RFS or HP-UX).

All solutions but the second require the ability to initialize the network interface from within the kernel, long before `init` had been spawned. In the early stages of development, this was deemed to be too complicated. Also, the third solution would require extensive modification of the file system code, since the code that 'mounts' the root file system needs a block device, and there was no easy way of initializing NFS from the boot code and mounting the root.

Rather, we selected the second solution. We wrote a device driver for a RAMdisk: a small (512K-1024K) amount of memory is preallocated in either the `.data` or the `.bss` segment, and initialized. The initialization is done either at kernel compile time or at boot time, respectively. In either case, we build the image of an AIX file system that fits in the amount of memory reserved for the RAMdisk, and load it as described above.

Our RAMdisk is implemented as a block device driver that statically allocates space from the `.bss` kernel segment and services block requests from there. It is roughly 200 lines of C code, and is fairly straightforward. The size of the RAMdisk itself is one megabyte, of which more than half is used by the startup scripts and programs (the rest is used as `/tmp` space.

The RAMdisk contains only the programs necessary to bring up AIX stand-alone (`/etc/init`, `/etc/inittab` and the shell), the programs to initialize the interface (`ifconfig` and `route`) and the `mount` command. It also contains miscellaneous scripts to automate some of the booting process.

When the system boots, it comes up in single-user mode. The user then runs one of the initialization scripts, which configures the radio interface, NFS-mounts the root directory of the designated server machine, make symbolics links for all the top-level directories (`/bin`, `/lib`, `/usr`, `/etc` and `/dev`) pointing to those on the server, and then brings up the system multi-user.

It should be evident by now that we are not trying to provide any stand-alone functionality. We *do* depend on the existent of infrastructure machines to make the SEN useable. Hence we put the smallest number of programs necessary on the RAMdisk itself, and put most of the initialization scripts and the entire filesystem tree on servers.

3.2.2 Swap Space

Ideally, we want our system to have enough memory so that no processes would swap, and configure the kernel so that it does not attempt to use swap or paging space. Paging/swapping is undesirable because it would have to be done over the network interface (as there is no local secondary storage) and at the current speed of the interface (128kbits/sec (see below)) transfering a 4kilobyte page would take a quarter of a second at best, which is unacceptably hight. We were unable to do this as the swapping/paging code was omnipresent. Although eight megabytes is not a lot of memory, it is enough to have a kernel, the RAMdisk, some server processes and an X server, a window manager and an X terminal, all in-core. When all these process are running, the amount of free memory is less than a megabyte. Additional applications run on infrastructure machines communicating with the SEN over the network and using its screen as a display.

We tried compiling the kernel with the swapping and paging code undefined, but that did not yield a runnable system. Then we noticed that when an AIX process is in core it is not using swap space (unlike, e.g., 4.2 BSD), and therefore it was not unreasonable to use main memory as our swap space. All we had to do was reserve more space for the RAMdisk than was needed for the root file system, and instruct the kernel to use that space by patching the appropriate kernel variables. By trial-and-error we determined that 1.5MB was enough swap space for the programs that run on the SEN.

Ideally, we want to swap over the radio interface. The options for doing this are similar to the ones given above for the root filesystem. With our current radio interface, this would be prohibitively slow, but this need not be the case when faster radio interfaces appear. In order to acquire some experience with network swapping, we shall be implementing a pseudo-device driver that appears to the kernel to be a block device driver (like a disk drive) but in fact forwards read and write requests to a user process communicating with a disk block server (à-la `nd`) on a server machine. The device driver provides for extensive buffering and has hooks for statistics gathering.

Paging statistics gathered this way can be very useful in deciding key parameters for future designs. They can be used in determining the amount of RAM necessary to keep

the paging rate low, in deciding which and how many applications should be run locally instead of on an infrastructure server, and so on.

3.3 Network Interface

The issue of networked operation for the SENs arose long before we even started working on the code. Given the fact that we were designing a *portable* machine to be used by students in classrooms, the library, the student lounge, etc., we could not limit its range by using permanently wired network interfaces (e.g., Ethernet or Token-Ring). Since the machine was to be able to use the infrastructure at all times, it was undesirable to design it so that everything is done locally, followed by a download/upload session to the user's home machine.

Clearly, all these constraints indicated that we should use wireless communications. We wanted the throughput of the wireless interface to be as high as possible, since communication with servers and other machines would be the rule rather than the exception, and that limited our choices of which interface to use. At the time the project started, the only available radio interface that was suitable was the Telesystem ARLAN-450 as discussed above.

The only software provided by Telesystem was Novell Netware ^{TM} software, software, that only worked under MS-DOS. Thus we would have to write our own drivers. Due to issues concerning proprietary information, we specified a set of low-level routines to control the device, and then we proceeded to write a device-driver for it. The other team had not UNIX expertise, but had a non-disclosure agreement for the specifications and were allowed to only give us object code.

The device itself can gather low-level statistics about its performance, and we added hooks in the driver to enable us to access those statistics. We also added the ability to save the first few (up to 64) bytes of each packet going through the device. We use these features to monitor our network (in "LANalyzer" fashion), and also gather statistics on the amount, distribution and quality of radio traffic. By 'quality' we man the fraction of packets that were successfully transmitted and received by the hardware, and not garbled or otherwise lost in transit. This is an essential feature in a radio network, where reception characteristics change as a function of time and physical location. The results of a survey done over a period of a few days are summarized in [BSLV90]. Briefly, we get an average data rate of 128 kilobits per second when using TCP connections, and the observed peak throughput is 158kbps. This is not using the full capacity of the channel, so two independent transfers can be going on simulateneously without major degradation of performance. If more than two transfers are active, then performance goes down as an almost linear function of the number of TCP connections. At the time of the writing of this paper, we do not have enough machines to conduct measurements of heavy-traffic behavior.

3.4 Display Support

The LCD display and the controller were chosen for their ability to provide a reasonably high resolution (640x480 pixels), fit in the SEN enclosure, and still be reasonably priced.

Although we had been told that the display controller was 100% VGA-compatible, it was in fact an EGA-like controller with what it called a "VGA-mode-12" (640x480x4 pixels) compatibility mode, which was quite misleading (it *did* have that resolution, but the programming interface was different).

Our first attempts at bringing up an AIX kernel up on the SENs were failing at just about the point where the kernel was taking control. Several weeks were spent trying to get the kernel (without the radio driver yet) running on the SEN. It turned out that we wasted a lot of time trying to find out how to set the parameters for the wrong device.

In any case, it turned out that the kernel was reinitializing the on-board VGA display controller, whose control and status registers (CSRs) conflicted with the CSRs on the LCD controller, and the kernel crashed. Progress was painfully slow. How does one debug display routines when one has no display to put diagnostic messages on? Answer: write special dot patterns on the screen to monitor the progress of the code.

An additional problem concerning the display is that the AIX initialization code uses two different terminal handlers which are inextricably meshed into the code and which provide support for nearly all known IBM terminals. This turned the development of our initialization process and X11 support into major tasks. Many of these problems resulted from the lack a well structured interface to the display hardware.

3.5 Stylus Support

Instead of a regular mouse, the SEN has a conductive stylus and a resistive screen overlayed on top of the LCD diplay. As discussed earlier in this document, this is to provide a paper-like interface to the user. While most of the work here is the domain of our user interface group, and most of the support code is in a modified X11R4 server, the user-level software still needed some operating system support.

The controller is connected to the serial port, and can thus be read from or written to as `/dev/tty0`. Standard (albeit undocumented in the AIX 1.1 documentation) System V terminal I/O processing could format the data we received from the controller (poll with a `DC1` control character, then get five bytes back – one byte for pen position, two bytes for x-coordinate, two bytes for y-coordinate). In order to present a better interface to the applications prorgrammer, and also to improve performance, we wrote a line discipline to replace the default line discipline which processes all of the characters for an input event in a single call.

The additional stylus support code is primarily a user-interface concern and is thus not further described here.

4 Applications

Needless to say, all AIX user programs can run on the SEN. In addition, all X11 programs can run, either locally or remotely using the SEN display. The entire suite of standard and user-contributed X11 programs is available to SEN users. In addition, the CMU-developed Andrew toolkit applications have also been ported to AIX. Students will be using applications like the Andrew multimedia editor (`ez`) to access and annotate texbooks, which have already been scanned and OCR'ed (with the permission of the

publishers). More applications are being developed to take advantage of the stylus and the touch screen.

5 Conclusions

To summarize our activity:

- We started with a rough set of specifications for the hardware.
- An independent group designed and built the prototype SENs from off-the-shelf components and custom-made enclosures. There was virtually no interaction between these two groups.
- We started with an off-the-shelf operating system, and added the necessary features to support our non-standard and exotic hardware
- We modified existing applications, and wrote additional applications for the unit. Finally,
- We are using fifteen SENs in a class as an experiment.

Our experience with the SENs taught us a number of things, which will be useful to us in designing the successor to the current prototype, and may also be useful to other people designing portable workstations:

- The decision to split the functionality between small portable units and larger infrastructure machines was the right one. It is still an open problem as to what functionality to put on each side.
- Wireless communication is, and will probably remain, severely limited in data rates (to increase the data rate we have to increase the operating frequency, and at higher frequencies the geographic coverage is much smaller). 2 Mbit/sec devices are just becoming available; this is comparable to the original 3 Mbit/sec ethernet but is only one fifth of current Ethernet speeds. Therefore, it seems that the key criterion in deciding upon "the split of functionality" will be the minimization of network traffic.
- In addition, protocols and higher-level applications should be designed with low expectations on the network throughput and with careful consideration of latency. For example, we need file systems with extensive caching, and high-level display software, (e.g., `XNeWS` instead of `X`) to minimize the network traffic.
- Early in the project we were trying to determine whether we should build a 'laptop', an X-terminal or a workstation. Using the criterion mentioned above, it appears that the decision to build something closer to a workstation was correct; it makes less demands on the network and the infrastructure than a plain windowing terminal, but is more powerful and usable than a laptop.
- Since a portable machine like the SEN has very few peripheral devices and enough memory to avoid paging or swapping, it makes use of a very limited subset of the operating system capabilities. Thus, we want to be able to trim the OS and only configure the pieces we are using. Most current OSs are too monolithic and non-modular to allow removal of unwanted components (other than optional device-drivers). For this reason, we are considering operating systems such as Mach release 3 (the 'kernelized' Mach) for our next OS.

- 640x480 pixels is barely enough screen real-estate. This is especially true if we want to display multi-font texts, graphics and have a paper-like interface. A megapixel flat panel display would be a start, with the goal being an A4-size (or 8.5x11 inches) display at 300 dpi. Although there are no such devices commercially available today, they are within reach of the state-of-the-art.

Porting an operating system to a new machine, especially one that it was never meant to run on, is always a significant undertaking. In our case, matters were complicated by nonexistent, incomplete or misleading documentation, by inexperience (only one of the students in the project had prior kernel- or systems-programming experience), and by less than optimal communication between the members of the group (due to inexperience in team-work of this magnitude). Despite all the problems, the prototype system was ready more or less on time, and is continuously being improved.

6 Acknowledgments

We would like to thank IBM Corporation for providing the financial support necessary for such and undertaking. Peter Bade and Barbara White of IBM-Milford for their cooperation in this project, David Bantz and his group at IBM-Yorktown for the low-level radio interface code, Håkan Winbom of IBM-Hawthorne for AIX-related support, and Jim Yarborough of IBM-Raleigh for designing and manufacturing the prototype SENs.

References

[BSLV90] Israel Ben-Shaul, Marios Levedopoulos, and Etienne Varloot. Arlan measurements. Technical report, Department of Computer Science, Columbia University, 1990. In preparation.

[CG85] W.J. Croft and J. Gilmore. Bootstrap protocol. RFC 951, September 1985.

[IM90a] John Ioannidis and Gerald Q. Maguire Jr. The coherent file distribution protocol. Technical Report CUCS-043-90, Department of Computer Science, Columbia University, 1990.

[IM90b] John Ioannidis and Gerald Q. Maguire Jr. Pip-1: A personal information portal with wireless access to an information infrastructure. Technical report, Department of Computer Science, Columbia University, 1990.

[SMIa] SMI. *Sun Unix Release 2.0 Documentation*.

[SMIb] SMI. *Sun Unix Release 4.0 Documentation*.