Minix Runs on the PT68K-2

By:
J. Gary Mills
1019 Weatherdon Ave.
Winnipeg, Manitoba, Canada R3M 2B5

About a year ago, I purchased a PT68K-2 from Peripheral Technology in Marietta, Georgia. I had been looking for a 68000-based microcomputer for some time, and they had come up with one that took advantage of the PC/XT clone components that were widely available. The PT68K-2 is a 68000 board that fits a PC/XT cabinet and has slots for IBM-compatible I/O cards. It has serial and parallel ports and a floppy disk controller, but lacks an MMU or a DMA controller. I ordered one with a megabyte of RAM, an IBM clone keyboard, and an IBM clone monochrome display card.

My PT68K-2 came with the 'Humbug' ROM monitor and the 'SK*DOS/68K' operating system, from Star-K Software Systems. SK*DOS looks like FLEX9, which was familiar to me because I ran it on my 6809 machine. Although SK*DOS is a fine system for many purposes, I was looking for something better. I wanted a unix-like operating system with a hierarchical file system, multi-tasking, and the familiar shell semantics. I had also grown a bit weary of system programming in assembler language. My search led me to consider Minix. The following information is an excerpt from the 'Minix Information Sheet', recently posted to comp.os.minix on USENET.

WHAT IS MINIX?

MINIX is an operating system that is a subset of UNIX Version 7. It contains nearly all the V7 system calls, and these calls are identical to the corresponding V7 calls. It also includes a Bourne-compatible shell, and close to 100 utility programs, including cc, grep, ls, make, etc. To the average user, it is effectively V7 UNIX. If you dig deep enough, you will, however, find some differences.

The MINIX kernel has been written from scratch by Dr. Andrew Tanenbaum <ast@cs.vu.nl>. It does not contain ANY AT&T code at all. The utility programs have been written by Andy Tanenbaum, his students, and a number of other people, including people on USENET. None of the utilities contain any AT&T code either. The shell, the C compiler, make, etc. have all been completely redone. As a result, this code is not covered by the ATT UNIX license, and it can be made available.

What CPUs does Minix run on?

MINIX was originally written for the IBM PC, XT, and AT. It has since been ported to the NS 16032 and the 68000 (Atari ST). It will also work on many 386-based machines.

How can I get Minix?

MINIX is being sold by: Prentice-Hall, Englewood Cliffs, NJ 07632 (1-800-223-1360), and Prentice-Hall Int'l, Hemel, Hempstead, England (+44 442 231555)

When ordering it, please specify one of the following versions:

MINIX for 640K IBM PC $79.95
MINIX for 512K IBM PC/AT $79.95 (0-13-583865-7)
MINIX sources on mag tape $79.95
MINIX code + reference manual (PC) $110 (0-13-584426-6)
MINIX code + reference manual (AT) $110
MINIX for the Atari ST $79.95 (0-13-584392-8)
Textbook: Operating Systems: Design and Implementation (0-13-637406-9)
Reference Manual: MINIX for the IBM PC, XT, and AT (0-13-584400-2)

How Can I Find Our More About Minix?

MINIX is described in detail in the following book:
Title:*Operating Systems: Design and Implementation*
Author:Andrew S. Tanenbaum
Publisher:Prentice-Hall
ISBN: 0-13-637406-9(Hardcover)
0-13-637331-3 (Paperback, outside of U.S and Canada)

A German translation was begun in Feb. 1988. There is also a paperback MINIX Reference Manual that is a subset of the book. It contains only the MINIX specific information, not the general background stuff on operating systems that the book contains The software package does not contain a manual; this is contained in the appendices to the book, which also contain a complete source code listing (in C) of the MINIX kernel.

Is Minix Public Domain?

No. MINIX has been copyrighted by Prentice-Hall. Prentice-Hall has decided to permit a limited amount of copying of the sources and binaries for educational use. Professors may make copies for students in their operating systems classes. Academic researchers may use it for their new experimental machines, and things like that. A small amount of private copying of diskettes for the use of personal friends is ok, but please do not make more than 3 copies from each original. Prentice-Hall is trying to be more reasonable than most software publishers. Please do not abuse this. Online repositories of the full source code distribution are not permitted. All commercial uses of MINIX require written permission from Prentice-Hall; for the most part, they are willing to grant such permission in return for a royalty on sales.

What Comes With Minix?

Minix includes the complete kernel source and binaries. Source for all commands except for the compiler and linker are also included. The following programs come with the Atari ST version:
ar as badblocks basename cal cat cc cdiff chmem chmod chown clr cmp comm compress cp cpdir date dd df diff diskcheck du echo expr factor false find fix fsck getlf grep gres head kill ln login lpr ls make megartc mined mkdir mkfs mknod more mount mv od passwd pr printenv pwd readall readfs rev rm rmdir roff sed sh shar size skdos sleep sort split stty su sum sync tail tar tee test time tos touch tr treecmp true umount uniq update uudecode uuencode wc

ADAPTATION TO THE PT68K-2

The Atari ST has the same 68000 cpu as the PT68K-2, so that version of Minix made a good starting point. However, the peripheral devices are almost entirely different. I borrowed an Atari ST and set to work, doing all the development under Minix, using the Minix compiler. All the kernel source files that were specific to the Atari ST, most of which were device drivers, had file names beginning with 'st'. The first task was to modify these files to suit the peripheral devices on the PT68K-2, creating an equivalent set of files beginning with 'pt'. This took about a month of evenings and weekends, and was made more pleasant by the nicely-written kernel code. Generally, only a small portion of each file needed to be modified. Once that was done, it only remained to build a kernel image and to create a boot disk for the PT68K-2. I won't claim that Minix booted on the PT68K-2 the first time, but it did boot and run the second time, after minor changes. The following section describes the hardware differences and the kernel changes that were required to create a version of Minix that would run on the PT68K-2.

Interrupt Handling

On the Atari, interrupts may be generated by the clock timer, the DMA device, the keyboard ACIA, or the parallel port. Interrupts from hardware are handled by a 68901 multi-function peripheral, which priorizes them and supplies a vector number to the cpu that invokes one of the first sixteen user vectors. Assembler code in the file 'stmpx.s' handles the vectored interrupts and calls interrupt service routines in various device drivers. The PT68K-2 has a more primitive interrupt system, using the non-68000-family interface. Interrupts may be generated by the clock timer, the IBM keyboard, or the parallel printer port. All interrupts from hardware are wired to IRQ5, invoking the level five autovector. For the PT68K-2, the file 'ptmpx.s' handles the one hardware interrupt and must poll status registers in various devices to determine which interrupt service routine to call. This file also reserves storage for 'shadow copies' of some registers in the peripheral devices. The reason for this is some multi-part devices, like the DUART or the PIT, have write-only registers that are shared between parts. Keeping shadow copies allows drivers for each part to be separate and not interfere with each other.

The Clock Timer

The Atari ST has a 2.4576 MHz clock which is divided by a programmable counter in the 68901 MFP to produce interrupts. The interrupt service routine in 'clock.c' does a further division by four to produce the 60 Hz clock tick used by the Minix scheduler and real time clock. Only minor changes were necessary to adapt the clock routines to the PT68K-2. A 3.6864 MHz clock is available to the first 68681 DUART, so that clock is divided by a counter programmed to produce interrupts directly at 60 Hz. No software division is required, resulting in a more efficient kernel. The timer in the 68230 PIT would have been a better choice, but it has no connection to the clock, and no interrupt line. Simple hardware modifications could remedy this. The interrupt should likely be at a higher priority than the keyboard interrupt.

The Keyboard

Changes to the keyboard driver were mainly a result of differences in the keyboard interface because both the Atari keyboard and the IBM clone keyboard transmit the same scan codes. The Atari ST uses a 6850 ACIA whereas the PT68K-2 uses a TTL keyboard register. The PT68K-2 keyboard register interrupts via an input line on the first DUART. Obtaining the scan code requires a read from one address to get the byte, followed by a read from a second address to reset the register. The IBM keyboard has built-in key repeat, so the software repeat routine in the Atari version is no longer needed. The file 'stkbd.c' also contained support for Atari national keyboards. This was deleted as well.

The Display

The Atari ST display is quite different from the IBM PC clone display card used in the PT68K-2. The Atari has a video controller device that uses 16 K of system RAM for a bit map of the screen. The driver copies information from font tables to form characters on the screen. On the PT68K-2, the video RAM and controller are on the display card. Each display position on the screen has a character byte and an attribute byte in video RAM. For the PT68K-2, the driver initializes the video controller registers to start the display with a blank screen. It does scrolling simply by copying bytes in video RAM, and does cursor movement by changing the cursor location registers in the controller. The font tables and associated code, of course, had to be deleted, but all the support for ANSI escape sequences was retained with only minor changes. The display driver also is responsible for the 'bell' tone, and on the Atari, it uses the sound device to generate the tone. On the PT68K-2, sound is produced by enabling and disabling an output from the first DUART that drives the speaker. Unfortunately, the timer in the DUART has to run at 60 Hz to serve as the system clock, but no other timer was available.

The DMA Device

The Atari ST uses a DMA device for access to the floppy disk and the hard disk, managed by routines in the file 'stdma.c'. The PT68K-2 has no DMA, so that data transfers to and from the disks must be done by cpu action. This is a basic limitation of the PT68K-2. The DMA routines are omitted from the PT68K-2 version of Minix, requiring corresponding changes in the floppy disk driver.

The Floppy Disk Driver

The Atari ST uses a Western Digital 1772 floppy disk controller, accessed via the DMA device. In the file 'stfloppy.c', the driver starts each floppy I/O operation by issuing a command to the controller. All operations interrupt on completion, so the interrupt service routine checks the result of the operation and takes appropriate action. The DMA device does the data transfers for sector read and write operations, also interrupting on completion. The floppy driver required considerable modification for the PT68K-2 version because, although the PT68K-2 also uses the WD 1772 FDC, it has no DMA, and the interrupt line is not connected. During sector I/O operations, the data transfer rate is too high to allow the cpu activity to be interrupted by other devices. It is therefore necessary to disable interrupts during these operations. Some interrupts are lost during sector I/O, affecting mainly the clock, but potentially also the keyboard and parallel port. The structure of the driver had to be revised to poll the FDC and wait for completion of each operation. Interrupts are enabled at this point, so other system activity can continue while the floppy driver waits. An attractive hardware modification would be to connect the FDC interrupt line and use interrupts to signal completion. The line should be a low priority interrupt, and would have to pass through a DUART or a PIT so it could be enabled by software when required. One advantage of doing this would be to allow a programmed time out to interrupt the FDC when accessing a drive with no disk inserted.

The Hard Disk Driver

The Atari ST has its own unique hard disk controllers. A driver could have been written to support the Western Digital controller card that the PT68K-2 uses, but the simplest adaptation was to defer this until later. Consequently, the file 'ptwini.c' is only a dummy hard disk driver, based on the Atari version.

The Printer Driver

Changes to the printer driver were mostly due to differences in the hardware. The Atari ST uses a parallel port in the 68901 MFP for a printer port. The PT68K-2 has a printer port on the IBM clone monochrome video card, but it is not usable because it has no interrupt line. However, the parallel port in the 68230 PIT is suitable. Interrupt handling is a bit tricky because the PIT will interrupt whenever the port output buffer is empty. In the file 'ptprint.c', the driver initializes PIT port A for pulsed handshake with interrupts disabled. The driver then only enables the interrupt when output is in progress and more characters remain to be output. This driver has not been tested, but will likely work.

Memory Size Determination

In the file 'mm/main.c', the Atari ST version of Minix reads a TOS variable to determine the memory size. The PT68K-2 version simply assumes that one megabyte of RAM is present. Minix would work with 512 K of RAM, so this could be changed to do a memory test of some sort.

Generic Kernel Files

There were many files under the 'h', 'mm', 'fs', and 'kernel' directories that contained code that is only compiled when the symbol 'ATARI_ST' is defined. These were all enhanced to produce the PT68K-2 version when the symbol 'PT68K' is defined. In many cases, only the symbol was changed, as the Atari code was also appropriate for the PT68K-2.

THE BOOT BLOCK FOR THE PT68K-2

The boot disk for Minix simply consists of a boot loader in the first sector, followed by the kernel image in consecutive sectors. It is conventially on a single-sided diskette. The task of the boot loader is to load the kernel image into memory and start execution. The Atari ST version of Minix used a BIOS call to do the load. For the PT68K-2, the boot block requires routines to drive the WD 1772 floppy disk controller for 'restore', 'seek', and 'read sector' operations. This code fits quite nicely into the 512-byte sector, leaving room for some variables required by Minix. The file 'bootblok.s' is included here as 'Listing 1'. To begin the boot, the Humbug 'fd' command loads the first sector into memory and jumps to the first location. Fortunately, Humbug has no problem loading a 512-byte sector, and the rest is done by the boot loader and the Minix disk driver.

```
! Boot block for the PT68K-2, complete with low level disk i/o
! for the WD1772.  Expects an 80-track single-sided disk in drive 0.
        .sect  .text
        .sect  .rom
        .sect  .data
        .sect  .bss

        .sect  .text
start:
        bra    boot          ! 000: jump to loader
        .ascii "MINIX "      ! 002: 6 byte identification
        .data1 0,0,0         ! 008: volume serial
        .data1 0,2           ! 00B: 512 bytes/sector (low byte first)
        .data1 2             ! 00D: 2 sectors/cluster
        .data1 1,0           ! 00E: reserved sector (low byte first)
        .data1 2             ! 010: number of FATS
        .data1 112,0         ! 011: number of dirs (low byte first)
        .data1 208,2         ! 013: 720 sectors (low byte first)
        .data1 248           ! 015: media descriptor (80 track SS)
        .data1 5,0           ! 016: sectors/FAT (low byte first)
        .data1 9,0           ! 018: sectors/track (low byte first)
        .data1 1,0           ! 01A: number of sides (low byte first)
        .data1 0,0           ! 01C: hidden sectors (low byte first)

! offsets in this boot block:
magic  = 502
nsect  = 504
fsckd  = 506
zero   = 508
fsckt  = 510

ldaddr       = 0x040000

! disk controller registers
comreg  = 0xFE0101
stareg  = comreg
trkreg  = 0xFE0103
secreg  = 0xFE0105
datreg  = 0xFE0107
dlatch  = 0xFE00C1

boot:
        move.w  #0x0001,d6     ! start with cyl 0, sec 1
        move.w  start+nsect(pc),d4
        move.l  #ldaddr,a3     ! load address in memory

read:
        tst.w   d4
        beq     rel
        bsr     dread
        bne     boot
rel:
        lea     copy(pc),a0
        lea     start(pc),a1
        sub.l   a1,a0
        add.l   #ldaddr,a0
        move.l  a0,0x0014
        divs    #0,d0          ! jump to copy routine in super state

copy:
        move.w  #0x2700,sr
        move.l  #8,a0
        move.l  #ldaddr+0x208,a1       ! start address of minix
```

```
    move.l #0x400,d0
cp2:  move.l (a1)+,(a0)+
    cmp.l  a0,d0
    bne    cp2
    add.l  #0x200,a0      ! skip tos variables
    add.l  #0x200,a1
    clr.l  d0
    move.w  start+nsect(pc),d0
    asl.l  #8,d0          ! multiply
    asl.l  #1,d0          ! with 512
cp3:  move.l (a1)+,(a0)+
    cmp.l  a0,d0
    bne    cp3
    move.l  ldaddr+0x204,a0
    jmp    (a0)           ! minix boot adres


dread:
    move.b  #0x20,dlatch   !side 0, dd, drive 0
    bra    dr2             !goto restore
dr1:
    bsr    sread           !read sector
    beq    dr3             !until successful
    add.w  #1,d3           !incr error count
    cmp.w  #10,d3          !until too many errors
    blt    dr1             !loop
dr2:
    move.b  #01,comreg     !restore
    bsr    wnbusy          !wait for completion
    clr.w  d3              !no errors now
    bra    dr1             !loop
dr3:
    add.w  #512,a3         !incr load addr
    add.b  #1,d6           !incr sector
    cmp.b  #9,d6
    ble    dr4             !if past cyl
    clr.b  d6              !reset sector
    add.w  #0x0101,d6      !calc next cyl
dr4:
    sub.w  #1,d4           !decr count
    bgt    dr1             !until all done
    rts                    !return

sread:
    move.l  a3,a2          !—>place for data
    move.w  d6,d7          !get next track sector
    move.b  d7,secreg      !give sector to fdc
    asr.w  #8,d7           !get track
    cmp.b  trkreg,d7       !if different track
    beq    sr1
    move.b  d7,datreg      !give track to fdc
    move.b  #0x11,comreg   !seek
    bsr    wnbusy          !wait for completion
sr1:
    lea    datreg,a0       !—>data reg
    lea    stareg,a1       !—>status reg
    move.b  #0x84,comreg   !read
    bsr    wait

sr2:
    move.b  (a1),d0        !check status
    btst   #1,d0           !drq?
    bne    sr3
    btst   #0,d0           !busy?
    bne    sr2
    bsr    wnbusy          !wait for completion
    and.b  #0x1C,d0        !mask errors
    rts                    !return

sr3:
    move.b  (a0),(a2)+     !get a byte
    bra    sr2             !loop

wait:
    clr.b  d7
wa1:
    sub.b  #1,d7
    bne    wa1
    rts

wnbusy:
    bsr    wait
    move.b  stareg,d0      !get status
    btst   #0,d0           !busy?
    bne    wnbusy          !loop
    rts                    !return with status
```

REQUIREMENTS FOR PT68K-2 MINIX

To run Minix on a PT68K-2, you need one megabyte of RAM, an IBM clone keyboard and monochrome display card, and at least one 80-track double-sided 3.5" floppy disk drive. It's not possible to use a terminal as the console because neither the Atari version nor the PT68K-2 version includes a serial port driver. You also, of course, need the Atari version of Minix, which comes with nine 3.5" diskettes and a 62-page manual. All diskettes except the 'boot' and 'tos' diskettes are usable on the PT68K-2.

RESULTS

Minix runs beautifully on the PT68K-2 - in some ways, better than on the Atari ST. It does, however, have limitations, and certain enhancements will likely require hardware modifications to the PT68K-2. It definitely feels like Unix. It's very solid. There are a few bugs, many of them reported on USENET, but just about everything works well, and works as expected. Having the source code for the kernel and the commands is a great advantage. When bugs are reported, and patches posted, it's very easy to apply updates and build a new binary. The emacs-inspired screen editor and the C-compiler work very nicely. Finally, because Minix is compatible with Unix, there are all those public-domain Unix source programs available, most of which will run on Minix with little or no modification. A programmer will feel right at home in this environment.