

# Linux for Embedded Devices

NCLUG 2006/08/08  
Stephen Warren

# Outline

- Product demo
- Product description
- Hardware overview/constraints
- Technology

# Product Overview

- “Digital Media Adapter” DMP3000
  - (Theoretically) Soon to be on sale
- Plays audio/video/images stored on a server PC
  - Custom server for Win32 only, also Linux internally
  - Also reads from USB storage
  - Memory sticks, hard disk etc.
- Sits in living room, Consumer Electronics UI
  - PC stays hidden in office/...
- Wired/wireless network connection
  - Proprietary network protocol:-)
- OS is Linux
- Custom media playback software

# Hardware Overview

- Pretty much a standard x86 PC
- AMD “Geode” CPU
- NVIDIA NFORCE2 chipset
- Regular NVIDIA graphics chip
  - NV17: Older, and cheaper!
- “DOM” (Disk On Module) for storage
  - Flash-based IDE-interface “hard-disk”

# BIOS

- Basically a standard PC BIOS
- Boot splash hides normal BIOS cruft
- Fast boot
  - e.g. don't initialize absent hardware
    - keyboard/mouse
    - floppy
    - ...
- Security for DRM
  - Won't allow F2/ESC to access BIOS
  - Won't boot from e.g. Network, USB HDD

# Boot loader

- We use LILO
- It just works
- Personal preference over Grub
  - Due to Grub doing silly things to me, such as:
    - mis-interpreting “xxxmem=1234” as plain “mem=1234” and refusing to boot due to lack of RAM
    - FC3 kernel upgrade wouldn't boot; installed LILO to fix!

# Kernel

- x86 architecture
  - Very easy to support; H/W is regular PC
- 2.6.10 kernel
  - Why? 2.6.9 was available when we first started making our own linux “distro”
  - Upgraded to 2.6.10 to get better USB support
  - It worked; stuck with it!
- Custom “.config” to save space
  - Important due to limited RAM and “HDD” space
    - So, turn off all the features we don't use
    - Almost no modules, for simplicity
- Some performance/feature patches
  - Details next

# Kernel patch - fbsplash

- Displays a graphical splash screen during boot
- Works like this:
  - LILO sets graphics video mode and boots kernel
  - Kernel command-line triggers fbsplash
  - lilo.conf:
    - vga=786 # VESA 640x480x24bit
    - append="splash=silent,theme:default"
  - initrd stores:
    - kernel fbsplash helper binary
    - fbsplash configuration file
    - JPEG image
  - Kernel runs helper to load image into frame-buffer early in boot sequence

# Kernel patch - squashfs

- Compressed filesystem
- Hence, read-only
- But fine for static application binaries etc
- Speeds boot time
  - Flash based HDD is slow
  - Compression means reading less data
  - CPU overhead of decompression is less than gain due to avoiding slow flash access
  - Bonus: Allows us to fit our s/w on the “HDD”!
- Simple tools to create FS images – easy to use

# Kernel patches – boot speed

- Boot time speedup
  - From “CE Linux Forum”:  
<http://www.celinuxforum.org/>
  - Shorten IDE delays
  - Remove RTC second boundary synchronization

# Userspace

- uClibc C library
  - A compact alternative to glibc
- Busybox for standard system binaries
  - A compact alternative to util-linux/fileutils/.....
  - ls, shell, modprobe, ... everything!
- Our own proprietary application

# Uclibc

- DOM is 64M; very space constrained
  - Must store two rootfs copies for upgrade safety
- glibc is huge (some might say bloated)
  - But, we didn't need its full feature set
- uClibc is a C library aimed at small size
  - <http://www.uclibc.org/>
  - And, configurable feature set, like the kernel
  - Save megabytes of HDD space
- libstdc++ works with uClibc just fine, for STL
  - Didn't investigate whether uClibc++ would work
    - Seemed immature when we looked – quite a while ago!
    - Recent releases might have changed this

# Buildroot

- <http://buildroot.uclibc.org/>
- Builds a custom toolchain and root filesystem
  - We only used it for the toolchain build
- What's buildroot for?
  - gcc knows about the C library
  - Can be persuaded to use non-default C library using (many, complex) command-line options
  - But, quite error prone to do that (lots of options to get right)
  - So, buildroot builds a uClibc targetted compiler

# Busybox

- <http://busybox.net/>
- All-in-one single binary that replaces many common command-line utilities;
  - sh, awk, grep, modprobe, mount, reboot...
- Aimed at space savings...
- Single binary -> lots of code sharing, which is how the space is saved
- Also, doesn't implement every feature of the utilities it replaces, but you won't miss them!

# Building FS/HDD images

- Filesystem images:
  - genext2fs
  - mksquashfs
- Hard disk images (including partitioning):
  - qemu

# genext2fs

- <http://genext2fs.sourceforge.net/>
- Creates a filesystem image
  - i.e. a single file containing a mountable filesystem
  - Input is a regular directory containing your files
  - Configuration file can:
    - Override permissions
    - Create device nodes
  - Don't need to be root to create device nodes or set file permissions, because that's just data in a file
  - Don't need write access to a real physical partition on which to run mke2fs etc.
  - Loop devices still need root/sudo/...

# genext2fs

- genext2fs \
    - b \$number\_of\_blocks \
      - l \$number\_of\_inodes \
        - d \$input\_directory \
          - D \$configuration\_file \
            - q \ # Squash owner/umask to 0:0/077
            - r 1 \ # Just one reserved block
- \$output\_filename

# mksquashfs

- <http://squashfs.sourceforge.net/>
- Exactly the same concept as genext2fs
- But, writes a file formatted in squashfs
- Same advantages!
- `mksquashfs \`
  - `$input_directory \`
  - `$output_filename \`
  - `-all-root \` # Force owner to 0:0
  - `-le` # Little-endian

# Bootable HDD image creation

- We could do this:
- fdisk a real hard-disk
- dd the filesystem images into the HDD
- Install LILO to the HDD
- That all requires root/sudo/...
- And a real hard-disk of the correct geometry
- Hence, doesn't lend itself well to automation on build machines with restricted access
- So, use an emulator, so we only need root *inside* the emulator, but can run the emulator as a regular user

# qemu

- We chose to use qemu – it just works
- <http://fabrice.bellard.free.fr/qemu/>
- Virtual hard-disks are backed by regular files
  - Explicit geometry is specified to match the real thing
- Empty HDD images are stored bzip'd, and decompressed each time qemu is run:
  - Created offline via “dd if=/dev/zero of=image”, then manual partitioning in qemu
- qemu booted using a “boot CD”, which contains the install script (details next page)

# qemu

- Boot CD is the same root filesystem that our target hardware uses – uClibc, busybox, ...
- Isolinux and mkisofs used to generate a bootable CD image
- Custom install script
  - kernel cmd-line: `init=/etc/install_script`
- qemu's networking support is used to mount a directory from the host (samba). The install script writes status information here for build script to determine whether the install script ran correctly inside qemu

# qemu

- mkisofs \
  - b isolinux/isolinux.bin -no-emul-boot \
    - boot-info-table -c isolinux/boot.cat \
      - J -l -o bootcd.iso -r input\_directory
- Qemu-0.7.2 \ # Things changed in 0.8
  - nographic \
    - hda hdd.img -hdachs 1015,3,42 \
      - cdrom bootcd.iso -boot d \
        - serial stdio \
          - user-net -smb status\_dir

# Questions?

Ask questions!