The Syslinux Project

8 February 2009

H. Peter Anvin
Intel Open Source Technology Center

FOSDEM 2009

Brief disclaimer...

Syslinux is not an official Intel® project, but the Intel OTC is a very cool place for me to work...

Legal Information

Current state of the Syslinux project

- Syslinux is a suite of bootloaders
  - SYSLINUX – FAT
  - PXELINUX – network
  - ISOLINUX – CD-ROMs
  - EXTLINUX – ext2/ext3
- Currently only for the x86/BIOS platform
  - This is due to a historical “core” in assembly language
  - Work is underway to fix this
- Sophisticated menu systems
- Extensible via extensive module API

Additional pieces

- MEMDISK – a disk emulator
  - Emulates a disk in memory
  - Allows booting of legacy “INT 13h” Oses, mainly DOS
- gPXELINUX
  - Collaboration with the Etherboot project
  - Enhanced capabilities for network booting while allowing standard PXELINUX features to be used
  - Supports HTTP, FTP, NFS, AoE, iSCSI in addition to TFTP
- isohybrid:
  - Allows ISOLINUX .iso images to boot from USB sticks

The x86 PC is an ancient platform

- First released in 1981 (IBM PC 5150); IBM AT 1986; PS/2 1987
  - Open platform, widely cloned
  - Most BIOS interfaces date from this era
  - Boot from floppy and hard disk only – in 510 bytes
- CD-ROM booting standard (“El Torito”) 1993
  - “Native” El Torito not widely supported until late 90's
- Network booting standard (PXE) 1997, revised 1999
  - Early releases were problematic
- USB drives now widely supported, but lots of bugs, still
  - However, there are tricks that help

A brief history of Syslinux

- 1994: Original SYSLINUX implementation
  - Overnight hacking binge (really...)
  - Designed for boot floppies
  - Needed to be small to fit, hence assembly
- 1999: PXELINUX – PXE network support
  - The PXE spec only allows 32 K for the Network Boot Program
  - SYSLINUX is already small, and users understand it
- 2001: ISOLINUX – “El Torito” CD-ROMs
- 2004: EXTLINUX – ext2/ext3 general purpose loader
  Extensibility via modular API, menu system
- 2006: Graphical menu system
- 2008: gPXELINUX, ISOLINUX “hybrid” support

Strengths of Syslinux

- Designed for dynamic systems
  - System discovery at boot time, not install time
- “Plays nice with others”
- Sophisticated user interface
- Modular API can implement UI, automation, new file formats...

Weaknesses of Syslinux

- Large assembly core
  - Core still includes filesystem driver and basic CLI
- Only available for x86/BIOS
- Does not support some exotic configurations
  - Dynamicity comes at a price...

gPXELINUX

- gPXE plus PXELINUX in one image
  - gPXE contains an extended PXE interface for PXELINUX
- Allows booting via HTTP, FTP, NFS, AoE, iSCSI, ...
- Intended to be a drop-in replacement for pxelinux.0
- Still needs a TFTP server for the initial bootstrap
- This can be skipped if NIC can be reflashed with gPXE

The Syslinux module API (“COM32”)

- Based on a small C library (klibc)
- Programming is very similar to normal userspace C code
- The main limitation is only sequential, readonly file access
- The most common modules:
  - User interface (e.g. menu systems)
  - File format modules
  - Policy modules
  - Diagnostic modules

User interface modules

- Currently two different menu systems:
  - Complex menu system – it does everything
  - Simple menu system – what most people use
- Graphics library makes the same code work for graphics, text, or serial console

  ... unless you use graphics-specific features

File format modules

- Add support for new loadable binary formats
- The module describes where in memory various bits go, then the Syslinux “shuffle” library puts everything into place
- Example: Microsoft SDI format
  - 199 lines total; 139 non-blank/comment lines
    - ... and much of what is left is error handling

Diagnostic modules

- A few modules are available to show hardware/BIOS state
- Most of this data is also available to modules

Policy modules

- Example:
  - Boot kernel X on a 64-bit machine
  - Boot kernel Y on a 32-bit machine with PAE
  - Boot kernel Z otherwise
- 129 lines, 70 non-blank/comment lines

  ... most of the rest was command line parsing

  (we didn't make X, Y and Z compile-time constants!)

And yes, you can combine them...

- There is *already* code in Syslinux to:
  - Probe your PCI bus
  - Map your devices to modules
  - Build an initramfs with the modules you need

    ... all on the fly

- The combinations, of course, are endless...

- But it never ends...

  ... boot devices on USB, Firewire, ... are harder to discover

Ongoing work

- Lua interpreter module
  - Allows scripts instead of module
  - Especially useful for policy modules
- Readdir support
- Move the filesystem code out of assembly
  - Really needed to support advanced filesystems like btrfs
- Switch rest of core from assembly and make it portable
  - The only left should be what is needed for BIOS support
  - This is required for EFI support to be possible
  - Early infrastructure work in progress

Core components

- First stage loader
- Disk/network I/O
- BIOS extender
  - Allows access to protected mode
- Shuffle system
  - Rearranges memory after loading
  - Must occupy minimal memory

NEEDS TO REMAIN IN ASSEMBLY

- Command line interface
- Config file parser
- Kernel loader/file parser
- Filesystem driver

SHOULD BE REWRITTEN IN C

Syslinux needs you!

- Syslinux is too large to be a one-person side project
- Syslinux now has a significant number of regular contributors

  ... but could really use more
- Newbie help and documentation is greatly appreciated
- Core rewrite will require a massive effort

  ALL OF THIS IS ONLY WORTH IT IF USERS WANT IT!

- User and distro feedback is highly valuable, too

Website/wiki

- **http://syslinux.zytor.com/**
- **Slides at:**

  http://syslinux.zytor.com/fosdem2009.pdf


intel®
Leap ahead™

Booting from disk

- The BIOS loads the disk Master Boot Record (MBR)
  - The MBR typically allows 424-440 bytes of code
- The MBR finds the bootable partition and loads its boot sector
  - The bootsector allows 510 bytes of code, except FAT
- The boot sector loads the *first* sector of the boot loader file
- ... which then loads the rest of the boot loader file
- From there, we can load modules, kernels, and anything else

- USB sticks are considered disks; however, some BIOSes have serious bugs in their handling. It's getting better, though.

Booting from CD-ROM

- The BIOS reads the “El Torito” tables to find the boot loader
- In theory, can load the entire boot loader
- In practice, too many BIOSes fail to load more than 2K bytes
- Thus, we load ourselves after the first 2K, again

With isohybrid, we have both the code for disk booting and for El Torito on the same image. It seems to work on most systems.

Booting from the network

- Here, the PXE firmware actually will load the entire bootloader
- However, the environment the bootloader finds itself in is quite different.
- PXELINUX does not rely on the firmware TFTP stack
  - The firmware TFTP stack has too many limitations
  - Unfortunately, this occasionally exposes bugs