Porting Linux
Embedded Linux Conference (Europe)

Porting Linux

About Jon Masters

- Been playing with Linux for 14 years (and the kernel for 13 of those), since the age of 13.
- Built embedded NMR scientific instruments, worked with Montavista UK, now at Red Hat.
- Author of the LKML Summary Podcast and the kernel column in Linux User & Developer.
- Co-author of Building Embedded Linux Systems (second edition) – O'Reilly (2008)
- My car still has an empeg :)

Porting Linux

Overview

- Why port Linux anyway?
- Background pre-requisities
- Early board work
- Bootloader bringup
- Initial kernel bringup
- Debugging
- Working with Upstream
- Trends

Porting Linux

Why port Linux anyway?

- Linux is very portable
  - Supports 23 architectures in the upstream “mainline” kernel tree of Linus Torvalds.
  - Kernel is mostly written in C, with some assembly (most architectures only need a dozen such files)
  - Split between high-level generic functions and low-level functions to abstract architectural differences.

Porting Linux

Why port Linux anyway?

- Linux is competitive
  - The number of Linux kernel developers contributing to the official kernel has tripled since 2005.
  - Feature growth continues with an average of 10K new lines of source code added every day.
  - In the hour you spend here 5.45 patches will on average be added to the upstream Linux kernel.

  - Source: Linux Foundation analysis

Porting Linux

Why port Linux anyway?

- Linux is cost effective.
  - A large amount of code to build upon.
  - Large (growing) community of developers.
  - I think we all know the rest.

Porting Linux
Background pre-requisities

- Hardware
  - Development board or simulator
    - Optional debugger, some kind of UART
    - Boards range in value from $200-$crazy
    - Implement the same architecture and platform as the final design but maybe with a number of hacks.
    - Simulator can be cheap and flexible (e.g. Android/OpenMoko/OLPC using QEMU).
    - See Pierre's talk on QEMU for more.

Porting Linux
Background pre-requisities

- Software
  - Toolchain (GCC, binutils, etc.)
    - PTXdist/crosstool/project specific
    - See Robert Schwebel's PTXdist talk.
  - Some kind of IDE
    - Likely to be Eclipse based, e.g. all the vendors.
  - You can get all of this from a vendor.

Porting Linux

Background pre-requisities

- Experience
  - Kernel development experience
    - Maybe not arch level, but at least driver work. Need to understand and study architectural issues.
  - Hardware reference documentation
    - Don't forget to check the errata (first!)
  - Books and resources
    - Some links later, also forums such as CELF.
  - Sign up to the Linux Kernel Mailing List
    - At least keep an eye on discussion. Don't miss topics like the ongoing generic-asm work by Arnd Bergmann.

Porting Linux

Early board work

- Test the board actually works
  - Write a simple LED flasher, print messages to the UART, have an idea that it does something.
  - If examples have been supplied by a board vendor, run them to make sure the board isn't defective.
- Test the debugger actually works
  - I've had hardware debuggers that would lose breakpoints, and other weirdness.

Porting Linux
Bootloader bringup

- Many Open Source friendly projects use U-Boot
  - Das U-Boot written by Wolfgang Denk, and maintained by many “custodians”.
  - http://www.denx.de/wiki/U-Boot
  - Supports ARM, AVR32, Blackfin, Microblaze, MIPS, NIOS, PowerPC, SH, and more.
  - Typically stored in on-board NOR or NAND.
    - Relocates itself into RAM, loads a kernel (and root filesystem in an initramfs).

Porting Linux
Bootloader bringup

- U-Boot Design Principles
  - "Keep it small"
    - A build of U-Boot with network support (if applicable) should fit in 256KiB.
  - "Keep it simple"
    - U-Boot should only do what is necessary to boot
  - "Keep it fast"
    - Get things running and then get out of the way.
  - "Keep it portable"
    - U-Boot is (like Linux) mostly written in C, with some assembly for unavoidable reset/CPU init/RAM setup/C stack environment setup.

Porting Linux
Bootloader bringup

- U-Boot is highly configurable
  - Many if (CONFIG_) conditionals
- Implementation split between “board” and “cpu”
  - Platform stuff under “board”, arch under “cpu”

Porting Linux
Bootloader bringup

- U-Boot “board” support
  - Linker script defining U-Boot image
  - boardname.c file with basic functions
  - (optional) assembly helper code if needed
  - Various functions the CPU code will call into
    - lowlevel_init
    - board_pre_init, board_init
    - checkboard
    - initdram
    - Testdram
    - get_sys_info, get_PCI_freq

Porting Linux

Bootloader bringup

- U-Boot “board” support
  - Board provided functions may be empty
  - The possible functions vary by supported architecture, documented in the source
  - Flash functions that end in _f
    - Callable before relocation into RAM is complete.
  - Relocated functions that end in _r
    - Callable only once relocation into RAM is complete.

Porting Linux
Bootloader bringup

- Implementing a new U-Boot “board” port
  - Use a similar board as a reference guide.
  - Start by bringing up the U-Boot prompt
  - Add some testing functions to exercise specific board features (another common use)
  - Later add drivers for additional devices
    - Ethernet, disk, flash parts, etc.
  - Become a custodian of your port
    - Custodians maintain their piece of U-Boot (usually in their own “git” tree) on the Denx git server.

Porting Linux

Bootloader bringup

- U-Boot “cpu” support
  - Much less common that you would need to port to an entirely new architecture
  - Typical system entry is in start.S
    - e.g. start440 for a PowerPC 440 system.
    - Initialize CPU cacheing asap (e.g. iccci/dccci)
    - Initialize CPU mode/context (e.g. SPRs)
    - Initialize MMU (e.g. no virtual/clear TLBs)
    - Provide interrupt and exception vectors
    - Setup minimal C stack environment
    - Finally end up in cpu_init/board_init

Porting Linux
Bootloader bringup

- Passing System Information
  - Historically, embedded Linux didn't have a direct equivalent of EFI/ACPI/Open Firmware.
  - Kernels were heavily bound to the specific board in question
    - recompile needed to set options
  - Kernel command line option passing was added
  - bdinfo structure on PowerPC
  - Recent work focuses on Flattened Device Tree.

Porting Linux
Bootloader bringup

- Flattened Device Tree
  - Expresses system information in the form of an Open Firmware style device tree
    - Location of system resources in physical memory map
    - Model and serial number
    - Installed and optional devices
  - Stored in a binary BLOB and passed to the kernel
    - Special utilities to convert text file OF-style trees
  - Linux can decode the fdt to figure out board info

Porting Linux

Initial kernel bringup

- Kernel Overview
  - Linux supports 32 and 64-bit systems of Little and/or Big Endian in nature.
  - Macros, wrappers, function pointers and common function names abstract away such differences.
  - The kernel is split into arch and platform code.
  - All stored under the “arch/” directory.
    - Each arch has flexibility into handling its own platforms

Porting Linux

Initial kernel bringup

- Kernel Overview
  - The “core kernel” includes the low-level arch support and high level functions
    - e.g. those in the top-level “kernel/” and “mm/” directories.
  - Other stuff (filesystems, networking, drivers) are not considered to be “core kernel”.
  - Source code overview
    - Use a tool such as LXR (lxr.linux.no) to browse.
    - Use a tool such as cscope (invoke it with cscope -kR) to search specific symbols.

Porting Linux
Initial kernel bringup

- Architectures
  - Live in “arch/”
    - Formerly also include/asm-archname
  - New architectures are rare
    - But several added this year alone (microblaze, S+Core).
    - Total in the official kernel is 23 today.
  - Typical mistake is to copy an existing architecture
    - Especially something wildly inappropriate, such as x86 for an ARM-like new architecture, complete with all of its (deprecated) system calls.

Porting Linux
Initial kernel bringup

- Architectures
  - The kernel tree has been known to have too much duplication (e.g. i386 vs. x86_64)
    - But it's being worked on, e.g. x86 unification.
  - Arnd Bergmann introduced generic-asm
    - A generic “ABI” that provides all of the core header functions needed by the higher level kernel code.
    - e.g. <asm-generic/atomic.h> provides atomicity functions such as atomic_add, including a generic version.
    - Also implementations of low-level mmu.h, mutex.h, pci.h, page.h and 121 other header files right now.

Porting Linux

Initial kernel bringup

- Architectures
  - Asm-generic used by several architectures already.
    - Especially the new S+Core architecture
    - Microblaze is the process of migrating
  - S+Core
    - Liqin Chen appeared on LKML several months ago with patches for a new (ARM-like) arch from Sunplus.
    - The architecture is a low power 32-bit RISC SoC, with a 32/16-bit hybrid instruction mode (Thumbish), optional MMU, optional DSP-like functions, user defined co-processors, 63 prioritized interrupts, SJTAG, etc.
    - Targeting: Home and Entertainment

Porting Linux
Initial kernel bringup

- Architectures
  - Arnd Bergmann reviewed the initial S+Core port
  - Sent many suggestions that Liqin dutifully followed.
    - Both gained from the experience.
    - Now a good reference architecture in S+Core
  - Only proposed a few months ago and already upstream due to good community interaction.
    - A success story and a role model.

Porting Linux

Initial kernel bringup

- Architectures
  - The S+Core tree:
    - boot/ - Target location for vmlinux.bin
    - configs/ - A defconfig example
    - Include/ - The “asm” directory. Many of the 89 files in here simply include their <asm-generic> counterpart. Some e.g. cache/VM bits, register specifics (threadinfo), etc. following the standard asm-generic ABI.
    - Kconfig - Standard kernel configuration data. There is also a debug verion of this file called Kconfig.debug.

Porting Linux

Initial kernel bringup

- Architectures
  - The S+Core tree:
    - kernel/ - The “head.S” low-level assembly entry point, irq.c interrupt bits, module.c ELF module loader bits, process.c bits specific to clone(), setup.c low-level bits for bootmem, sys_call_table, sys_core, and time.c.
    - lib/ - Various low-level implemenations of things like strlen written in fast assembly.
    - Makefile
    - mm/ - pgtable.c, init.c (paging_init and mem_init), tlb-miss.c, tlb-score.c, etc.

Porting Linux
Initial kernel bringup

- Porting to a new architecture
  - Get to know the kernel tree first.
    - LXR, cscope, and others are your friends.
  - Pick an existing similar (endianness, bit size, behavior, etc.) arch and look at its implementation.
  - Don't copy an existing architecture.
  - Create your new one and pull in the asm-generic bits. Look to S+Core (“score”) and eventually to Microblaze for good example code.
  - See also: Nina Wilner's PowerPC presentation

Porting Linux
Initial kernel bringup

- Typical init process
  - Read through the code beginning with the head.S entry for your favorite reference architecture.
  - head.S
    - Conventional name for lowest level entry (usually at “start”, “_start”, “start_here”, or similar)
    - Entered directly after U-Boot exec.
    - Responsible for early configuring the CPU
      - Cacheing, initial stack sufficient for C code, enbale (SW/HW) MMU, jump to core kernel start_kernel
    - And providing exception vectors
      - Errors, Faults (page faults), etc.

Porting Linux
Initial kernel bringup

- Typical init process
  - start_kernel
    - Sequentially initialize the kernel.
    - Initialize lockdep/stack canary
    - boot_cpu_init. Activate the first processor using hotplug.
    - setup_arch. Architectural specifics. For example:
      - Low-level CPU and platform init
      - Paging (VM) enabled
      - Data passed in from the bootloader (device tree)
      - On S+Core: cpu_cache_init, tlb_init, bootmem_init, paging_init, and resource_init.
      - On PowerPC: enabling xmon debugger and debug output.

Porting Linux

Initial kernel bringup

- Typical init process
  - start_kernel
    - setup_command_line. Use the bootmem allocator to stash away the kernel command line.
    - sort_main_extable. Sort the kernel symbol table for later use by the module loader (recent speedup work here by Alan Jenkins and also Carmelo's LKM fast loader later).
    - mm_init. Calls arch-specific mem_init, sets up various kernel caches and enables vmalloc.
    - sched_init. Does the heavy lifting to prep the scheduler (allocating using bootmem the runqueus and CFS bits).

Porting Linux

Initial kernel bringup

- Typical init process
  - start_kernel
    - early_irq_init. Allocate the IRQ structs.
    - init_IRQ. Architectural counterpart to early_irq_init, providing platform specific stuff.
    - timekeeping_init. Generic function that determines which clocksources to use and configures them.
    - time_init. Corresponding architectural specifics.
    - console_init. Enables the console to that we can begin to output the various kernel boot messages.
    - kmemleak_init. Initialize Catalin's nifty leak detector.

Porting Linux
Initial kernel bringup

- Typical init process
  - start_kernel
    - calibrate_delay. Determine the “bogomips”.
    - fork_init. Prepare to be able to fork (clone) new tasks. Calls down into the arch code to complete this.
  - rest_init
    - Prepare the scheduler (including RCU)
    - Start the master kernel thread (kthreadd)
    - Setup the idle task and schedule into init
    - After that heading toward userspace

Porting Linux

Initial kernel bringup

- Platforms
  - As with U-Boot, platforms build upon architectures.
    - PowerPC implements a clean “platforms” directory.
    - ARM mixes things around under the CPU type.
    - Others (such as x86) don't really handle many different (non-PC) platforms all that well (yet).
    - Some platforms use structs of function pointers
      - PowerPC uses a define_machine macro, including a probe function that can selectively utilize the device tree.
      - ARM uses a MACHINE_STARTS macro, but is not yet as flexible. For example, board-n8x0.c registers n8x0_init_machine to be called for the Nokia N8xx tablet initialization.

Porting Linux
Initial kernel bringup

- Platforms
  - Platform devices
    - Many platforms are built using standard parts such as PCI (or PCI like) devices that can be registers and managed generically.
    - Some “devices” are connected to legacy buses or aren't really on a traditional bus at all
      - As is the case for many mapped SoC devices.
    - The Linux driver model documentation (in the “Documentation/” kernel directory) will show you how to register and manage platform devices
      - Needed for power management.

Porting Linux

Initial kernel bringup

- Porting to a new platform
  - This is far easier than porting to a new arch, since it's just a variant.
  - Typically, you can base your platform port on an existing platform for the arch in question and more legitimately copy/paste where not generalizable.
  - Make sure you educate the kernel about system geometry (RAM size, etc.) and location of PCI.
  - Use the platform abstraction for any generic mapped devices not managed elsewhere.

Porting Linux

Device Drivers

- Basic architecture and platform support have little meaning without drivers for peripherals.
- Fortunately, Linux already supports a large (growing) number of existing devices that may already cover the majority of your design.
- Refer to Linux Device Drivers (3rd edition) for more information about writing drivers.

Porting Linux
Debugging

- Many debugging and diagnostic options.
  - gdb. Can be used to attach to a remote hardware (or virtual machine) gdbstub and issue instructions.
  - ftrace. An in-kernel function tracing framework, originally used to measure kernel latencies.
  - kexec/crash/kdump. Can be used to boot an aux. kernel if the main one crashes, to capture state. A recent enhancement allows “flight recorder” mode.
  - Ksplice. Dynamically patch your running kernel.
  - Performance Events (“perf”). Capture system performance metrics (and almost anything else).

Porting Linux

Working with upstream

- Why you need upstream
  - Less “bitrot” due to constantly evolving upstream kernel. Reduces “rebasing vs. retaining” tradeoff.
  - More influence on future development. People will care about your project if it has code upstream.
  - 70% of total contributions to the kernel come from developers working at corporations that consider such participation a competitive edge.
  - Source: Linux Foundation analysis.

Porting Linux
Working with upstream

- Development Trees
  - The official kernel lives in Linus Torvald's “git” tree on git.kernel.org
  - There are countless other “git” trees available.
    - linux-next is a stepping stone
      - Stephen Rothwell posts a new tree each day
      - Made from 140 “git” trees that are merged
    - staging is for immature code
      - Lives in a special kernel directory (harder for arches)
      - Greg Kroah-Hartman periodically updates it
  - Please read Documentation/development-process

Porting Linux
Working with upstream

- Where do I go from here?
  - Check the MAINTAINERS file to see who owns the architecture or other kernel subsystem concerned.
    - Reach out to the community for advice if unsure.
  - Learn to use “git”, “quilt”, and the git email features.
  - Consider the “staging” tree for immature code.
  - Prepare your work for linux-next.
    - Track Stephen Rothwell's tree regularly and post a “git” tree of your patches.
    - Code that passes review and is in linux-next has a very good chance of being merged upstream in the next “merge window” by the relevant maintainer.

Porting Linux
Working with upstream

- Mailing Lists
  - http://vger.kernel.org
  - LKML – Linux Kernel Mailing List
  - Linux-next Mailing List
  - Architectural Maintainer Lists
  - Greg Kroah-Hartman's Driver Development List
  - etc.

Porting Linux
Trends

- Boot time
  - Work is going on in boot time reduction. See the talk today and upstream “bootchart”/“timechart”.
- Dynamic Power Management
  - Rafael J. Wysocki implemented dynamic suspend of individual buses in a recent rework.
- Flattened Device Tree
  - Continued work is happening here. Thanks to Grant Likely and others for their efforts.

Porting Linux
Links

- LWN - http://www.lwn.net/
- LKML - http://vger.kernel.org/
- Understanding the Linux Kernel
- Linux Kernel Development
- Linux Device Drivers (LDD3)
- Building Embedded Linux Systems
- Linux Kernel in a nutshell

Porting Linux

Disclaimer

- I do not speak for my employer.
- Web: http://www.jonmasters.org/
- Email: jcm@jonmasters.org

Questions?

Porting Linux
Trends

- Devtmpfs
  - Devfs v2.0? Not quite.
- IO Bandwidth Limiting
  - Several proposals (dm-ioband, IO scheduler) but nothing agreed on just yet.
- Swap
  - Compcache. Compressed RAM alternative to swap.

Porting Linux
Trends

- Virtualization
  - Various work to implement low-overhead (even low-latency “Real Time”) enhancements in KVM.
  - KSM. Kernel Shared Memory allows dynamic sharing of identical pages and is just one cool technology recently pulled into KVM.