# Embedded Linux: Systems and Software

***Jon Sevy***
***Geometric and Intelligent Computing Lab***
***Drexel University***
***May 2008***

# Table of Contents

# Embedded Linux Systems Overview

# Embedded Linux Systems Overview

- Components
- Kernel
- Libraries
- Applications
- System initialization and scripts
- Root filesystem
- Runtime Linux System
- Kernel space vs user space
- Virtual/physical memory
- Development system requirements

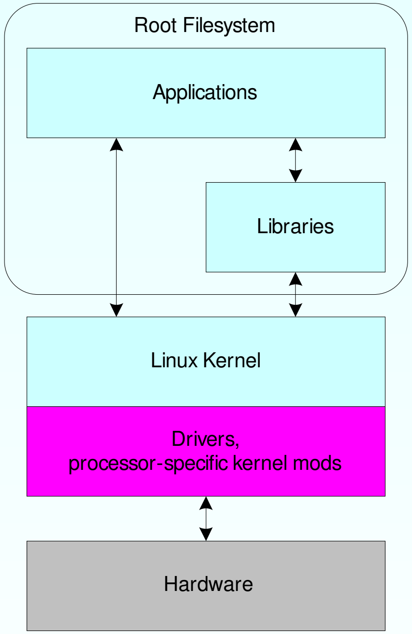
- Activities
- Resources

# Embedded Linux

- Any small system running Linux
  - “Headless” (no display – wireless router, set-top box, e.g.)
  - User-interactive (PDA, cellphone, etc.)
- More than just kernel!
  - Applications provide system-specific functionality
  - Shared libraries support applications
  - Kernel manages running applications, hardware drivers
- Think of as stripped-down desktop system
  - Unneeded features removed
  - Embedded-specific features added

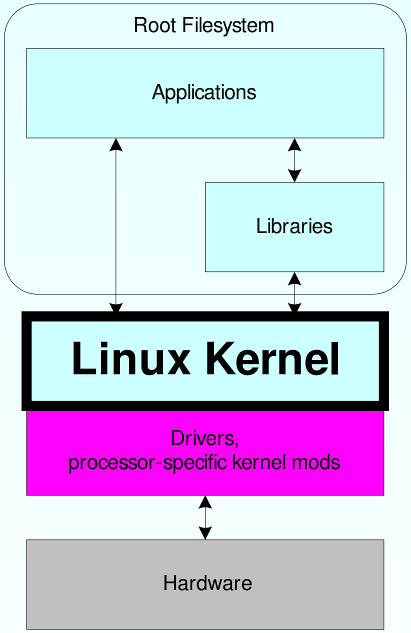
# Linux Software System Components

- Kernel
  - Manages tasks, drivers
- Drivers
  - Manage hardware resources
- Root filesystem
  - Libraries
  - Applications (including GUI)
  - Scripts
  - User data



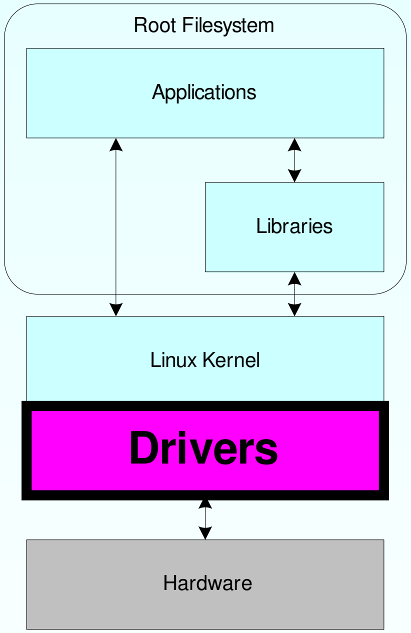
# Kernel

- Current Linux kernel: 2.6 series
  - Fully supports ARM processors (including ARM926)
  - Complete networking, filesystem, other support
- Configurable
  - Build in only those features needed
- Multiple possible execution modes
  - Execute-in-place (XIP)
  - Compressed/loadable



# Drivers

- Manage hardware resources (peripherals)
- Exist for many standard peripherals
- Built-in to kernel or loadable at run-time
- Well-documented process for creating custom drivers (see references)

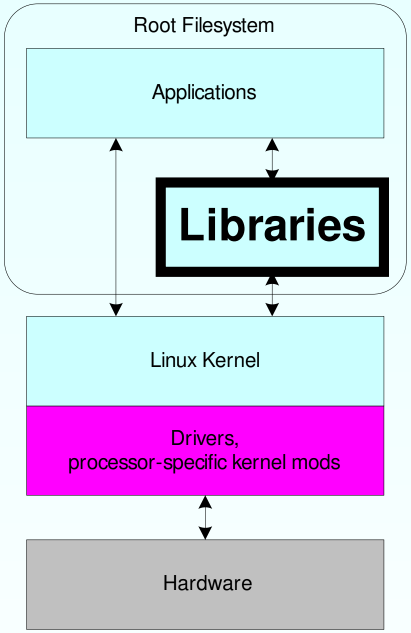


# Root Filesystem

- Directory tree containing needed libraries, scripts, applications
  - Organization usually follows standard Unix filesystem conventions (/bin, /sbin, /etc, etc.)
- Stored as standard Linux filesystem type
  - Typically cramfs or jffs2 compressed filesystem when in Flash
  - Ext2/3 for disk

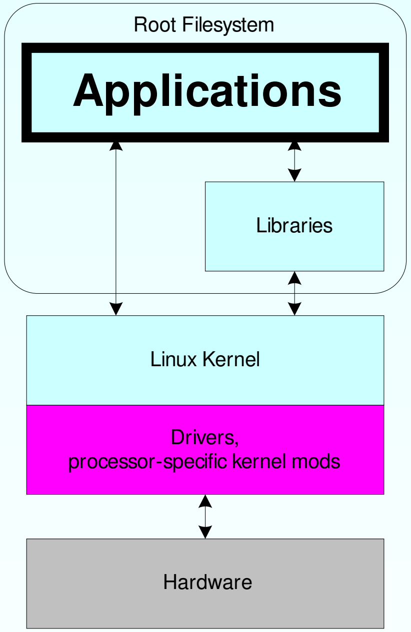
# Libraries

- C library
  - Standard utility functions, interface to kernel functionality
  - Several variants:
    - Glibc: big and full-featured
    - uClibc: small, configurable, targeted for embedded systems (usual choice)
- Others as needed
  - Pthreads
  - ALSA
  - GUI support



# Applications

- Created as standard Posix/Unix applications
- Stored in filesystem, loaded to RAM for execution
- Standard applications
  - Busybox
    - Standard Unix utilities in single package
    - Configurable feature support
- Custom applications
  - GUI applications
  - Anything system-specific (background network applications, etc.)



# Scripts

- Used to initalize/shut down system
- Others for access control, configuration
- Stored in /etc directory of root filesystem

# GUI

- Provide desktop environment
  - Window environment for GUI application creation and management
  - Many standard apps available (productivity, multimedia, etc.)
- Qtopia Phone Edition
  - Commercial, royalty-based
  - Complete suite of applications
  - Used in existing handset designs
    - Motorola A760, A780
    - Philips Nexperia Cellular System Solution 9000 reference platform

# Runtime Linux System

- Serial console
- Apps started at system initialization
- Daemons (always running services)
- Kernel threads (e.g., JFFS2 garbage collection)

# Memory Considerations

- Kernel space vs user space
  - MMU enforces protection
  - Requires copy or MMU map (mmap) to exchange data
- Virtual memory addresses
  - Application address space (0x0)
  - Kernel address space (0xC000 0000)
  - I/O address space (0xF000 0000)
  - /dev/mem, /dev/kmem, devmem2
    - Driver interface to inspect memory, used by devmem2/peek-poke

# Activity and Resources

- Activity
  - Skulk around an embedded Linux system
  - Use devmem2 to inspect memory
  - Use ps, top to see running system info
  - cat some /proc files to get kernel info

- Resources
  - Building Embedded Linux Systems, Karim Yaghmour, O'Reilly
  - Embedded Linux: Hardware, Software and Interfacing, Craig Hollabaugh, Addison Wesley

# Creating, Configuring and Building Embedded Linux Software Systems

# Creating, Configuring and Building Embedded Linux Systems

- Kernel
- Libraries
- Applications
- System initialization and scripts
- Root filesystem
- Loading on target

- Activities
- Resources

# Kernel - Configuration

- Acquiring source
  - http://www.kernel.org
  - full ARM support standard

- Configuring with menuconfig
  - make menuconfig ARCH=arm
  - built-in vs loadable modules: y vs m
  - .config/config.h and defconfig files
  - command line: root=/dev/mtdblock2 rootfstype=jffs2 console=ttyS0,115200 init=/linuxrc
  - asm -> asm-arm and arch -> arch-vx115 after configuration

# Kernel - Building

- CROSS_COMPILE environment variable in top-level Makefile
  - Set to prefix of toolchain; arm-none-linux-gnueabi- for CodeSourcery toolchain
  - Can set on command line or as environment variable
- make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
  - zImage: in arch/arm/boot
    - Self-extracting compressed kernel
  - loadable modules: in .tmp_versions
    - can install into root filesystem with correct subdirectory structure with modules_install and INSTALL_MOD_PATH:

      make modules_install INSTALL_MOD_PATH=../rootfs/rootfs

# C library: uClibc or glibc

- uClibc
  - configuring with menuconfig:
    - make menuconfig
    - need to set cross-compilation setting
  - building
    - make

- glibc
  - can use binary from toolchain
  - can configure and build with configure and make (see next)

# Other Libraries

- Typical library (e.g., ALSA)
  - configuring with configure:
    - ./configure  <options>
    - sets up files for building (may create Makefiles, configuration headers)
  - finding/setting options
    - ./configure –help
    - target:
      - target=arm-none-linux
    - cross-compiler
      - CC=arm-none-linux-gnueabi-gcc  as configure option, or
      - export CC=arm-none-linux-gnueabi-gcc; ./configure <other-options>
  - saving command for later (in config.log)
  - config.cache (may need to delete between reconfiguration)

# Applications

- Busybox
  - bundles most needed Unix apps
  - configuring with  make menuconfig
  - building with make

- Other (e.g., ALSA utils)
  - configuring with configure
    - may need to add CFLAGS, LDFLAGS variables with paths to needed headers and libraries (e.g., ALSA lib)
  - building with make

# Scripts and Initializations

- linuxrc
  - first user code run by kernel; specified in kernel command line (init=linuxrc)
  - does some basic filesystem mounting, etc.
- init.d and rc2.d directories and links
  - shell scripts to start/stop services in init.d
  - arg to each will be start, stop, restart
  - links to scripts in rc2.d, executed by init
- init
  - runs scripts in /etc/rc2.d directory for system service startup and shutdown
  - scripts starting with 'S' run at startup with argument “start”
  - scripts starting with 'K' run at shutdown with argument “stop”
  - scripts run in lexical order (hence numbers in names)

# Root Filesystem

- Create tree on development host
  - create required directories as part of build process
- Populate with apps, libraries and scripts
  - /dev: use mknod to create device nodes
  - links to RAM disk for /tmp, /var for Flash-based systems
- Package as filesystem for loading on target
  - use mkfs variants to create binary filesystem object (e.g., mkfs.jffs2)
- Loading on target
  - Create srecs using objcopy, load to Flash

# Activity and Resources

- Activity
  - Configure kernel, uClibc, Busybox
  - Configure and add an open-source library to distribution
  - Configure and add an open-source application to distribution

- Resources
  - Building Embedded Linux Systems, Karim Yaghmour, O'Reilly.
  - Embedded Linux: Hardware, Software and Interfacing, Craig Hollabaugh, Addison Wesley.
  - Busybox: http://www.busybox.net
  - uClibc: http://www.uclibc.org/

# ARM Linux Boot Process

# Linux Boot Process

- Bootloader requirements
- zImage decompression
- Kernel code
- System initialization

- Activities
- Resources

# Bootloader Requirements

- Virtually none if use head-<mach>.S to set machine/arch numbers
- Can pass tag structures to kernel for configuration
- Can use bootloader (uboot, blob, ...) to read kernel zImage from filesystem if desired

# zImage Decompression

- arch/arm/boot/compressed/head.S
  - include arch-specific code
    arch/arm/boot/compressed/head-<mach>.S
  - decompress kernel to RAM
  - jump to start of kernel in RAM (zreladdr)
    - zreladdr = ZRELADDR = zreladdr-y
    - zreladdr-y specified in arch/arm/mach-<mach>/Makefile.boot

- arch/arm/boot/compressed/head-<mach>.S
  - added to build in arch/arm/boot/compressed/Makefile
  - linked into head.S by linker section declaration: .section “start”
  - flush cache, turn off cache and MMU, set machine and arch number

# Kernel Code

- arch/arm/kernel/head.S: stext
  - look up machine and arch structures
  - set up initial kernel page tables, init MMU
  - copy data segment, zero BSS
  - jump to start_kernel

- init/main.c: start_kernel
  - initialize subsystems and built-in drivers
  - start init process

# Resources

- Linux Kernel Cross-Reference
  - hypertext-linked browsable kernel source
  - http://lxr.linux.no/

# Linux Board Port

# Linux Board Port

- Machine and processor ID
- Memory configuration
- Flash configuration
- Kconfig and Makefile modifications
- Platform includes: include/asm-arm/arch-xxx
- Platform source files: arch/arm/mach-xxx
- Interrupts
- Serial/console driver

- Activities
- Resources

Note:

- Use port to an ARM-based processor vx115 and platform vx115_vep development board as example

# Machine and Processor ID

- Machine and processor ID
  - arch/arm/tools/mach-types
    - define machine and arch numbers and macros
    - arch/arm/Makefile
    - machine-$(CONFIG_ARCH_VX115) := vx115

- Boot files
  - arch/arm/boot/compressed/head-vx115.S, Makefile
    - flush cache, turn off cache and MMU
    - set up machine and arch numbers

# Memory Configuration

- include/asm-arm/arch-vx115/memory.h
  - #define PHYS_OFFSET 0x24200000
    - physical address of kernel code base
  - #define PAGE_OFFSET (0xc4200000UL)
    - virtual address of kernel code base
  - #define MEM_SIZE 0x01e00000
  - used in virtual-physical memory translation functions
  - replaced by defines in discontiguous memory file if needed
- arch/arm/Makefile
  - textaddr-$(CONFIG_ARCH_VX115) := 0xc4208000
    - kernel entry point (virtual); address of stext in link map (vmlinux.lds)
- arch/arm/mach-vx115/Makefile.boot
  - zreladdr-y := 0x24208000
    - physical address where decompression routine jumps when done
- arch/arm/mach-vx115/vx115_vep.c
  - .phys_ram = 0x24200000 in MACHINE_DESC struct
    - start of RAM for use by kernel

# Platform-Specific Directories

- include/asm-arm/arch-vx115
  - contains platform-specific header files
    - hardware.h, others
  - configuration process generates symbolic links
    - include/asm -> /include/asm-arm
    - include/asm/arch -> /include/asm-arm/arch-vx115
- arch/arm/mach-vx115
  - contains platform-specific source files
    - main board files (vx115_vep.c)
    - interrupt, DMA, other SoC-related files

# Platform Includes: include/asm-arm/arch-vx115

- Required headers
  - hardware.h
    - platform hardware register defines
      - note use of virtual register addresses
    - included into arm generic hardware.h (include/asm-arm/hardware.h)
  - system.h
    - define arch_idle, arch_reset functions to indicate behavior when idle or on reset
  - dma.h
    - define MAX_DMA_ADDRESS to indicate all of memory is DMA-able
  - io.h
    - define IO_SPACE_LIMIT to mark all memory as possible I/O space
  - timex.h
    - define CLOCK_TICK_RATE, used in jiffies.h for system timing params
  - param.h
    - define HZ to set kernel tick rate different from 100/sec if desired

# Platform Includes: include/asm-arm/arch-vx115

- Required headers (cont.)
  - serial.h
    - used to put in standard (8250) serial port defines if using these
  - system.h
    - define arch_idle, arch_reset functions to indicate behavior when idle or on reset
  - vmalloc.h
    - some memory allocation defines
    - moved to common kernel code in 2.6.18 since same in all platforms
  - uncompress.h
    - output routines for zImage decompression stage
  - entry-macro.S
    - very low-level interrupt handling (described below)

- Other headers
  - anything hardware-ish

# Platform Source Files: arch/arm/mach-vx115

- vx115_vep.c
  - main board-specific initialization file
  - I/O mapping
    - define I/O virtual-physical map in map_desc struct array
    - define map_io function for MACHINE_DESC struct
  - Interrupt initialization
    - define board-specific irq_init funtion for MACHINE_DESC struct
  - Device specification
    - define platform_device and amba_device structs for use in driver configuration
  - Machine initialization function
    - vx115_init_machine
    - Register devices; will be matched with appropriate drivers for driver configuration

# Platform Source Files: arch/arm/mach-vx115

- vx115_vep.c (cont.)
  - Fixup function
    - set memory bank info
  - MACHINE_DESC struct for platform
    - pointers to platform functions defined above, and system timer
    - linked into list of supported machines; retrieved during boot

# Platform Source Files: arch/arm/mach-vx115

- irq.c
  - define functions to ack, mask, unmask irqs
  - define irqchip struct
    - function pointers for irq ack, mask, unmask
  - define irq initialization function
    - initialize controller, handlers to use specified irqchip

# Platform Source Files: arch/arm/mach-vx115

- time.c
  - System timer: define platform timer tick function
    - just manages hardware timer, calls system timer_tick function
  - define initialization function, sys_timer struct for use in MACHINE_DESC macro

# Platform Source Files: arch/arm/mach-vx115

- Others
  - other board-specific source files
  - gpio.c
    - gpio interface
  - dma.c
    - dma controller driver
  - ssp.c
    - ssp driver; probably belongs in drivers/char

# Kconfig and Makefile Modifications

- Have Kconfig and Makefile in each subdir

- Kconfig
  - add selectors for defines in code and Makefiles
  - defines generated into .config (also config.h for code header)

- Makefiles
  - add to lists of files to be compiled and linked into that subdir's objects
  - obj-y: built-in code file list
  - obj-m: loadable module file list

# Interrupts

- include/asm/arch/entry-macro.S
  - defines assembly routine get_irqnr_and_base
  - returns the IRQ number from controller

- arch/arm/mach-vx115/irq.c
  - defines irq mask/ack routines (discussed above)

- common kernel routines
  - arch/arm/kernel/entry-armv.S
    - low-level assembly vector handling
    - calls machine-specific get_irqnr_and_base and common asm_do_IRQ
  - arch/arm/kernel/irq.c
    - asm_do_IRQ: (eventually) calls IRQ-specific handler

# Flash Configuration

- drivers/mtd/maps/vx115_flash.c
  - define map_info struct indicating parameters for Flash devices (base addr, bank width)
  - define mtd_partition struct for each bank giving logical partitions
  - define init_vx115_vep_flash function
    - register flash map and partition info
    - module_init macro places function pointer in init section so will be called during system initialization

# Serial Console Driver

- provide serial driver for kernel control
  - complex structure; routines for input/output and control through ioctl's
- specify console on kernel command line
  - console=/dev/ttyS0

# Extracting Changes: Diff and Patch

- Pull out changes so can be applied to vanilla kernel
  - can deliver just changes rather than whole kernel
- Use diff and patch
  - diff: find all differences
  - patch: apply differences to fresh kernel
- Creating patch
  - need both unmodified source tree and modified source tree directories
  - diff -Nur (unmodified-source-dir) (modified-source-dir) > mods.patch
  - extra args to adjust diff process
    - --exclude=CVS
    - -I '.*$Id:.*' -I '.*$Id$.*' -I '.*$Revision:.*' -I '.*$Source:.*' -I'.*$Date:.*' -I '.*$Header:.*' -I '.*$Author:.*'
- Applying patch
  - from within top-level directory of “vanilla” source tree
  - patch -p1 < mods.patch

# Activity and Resources

- Activity
  - work with diff and patch

- Resources
  - Porting the Linux Kernel to a New ARM Platform (2.4-series kernel): http://linux-7110.sourceforge.net/howtos/netbook_new/porting2arm_aleph.pdf
  - Linux Porting Guide (uses MIPS as example): http://www.embedded.com/shared/printableArticle.jhtml?articleID=9900048

# Linux Device Driver and Kernel Programming

# Linux Device Driver and Kernel Programming

- Device and Driver Model
- Loadable vs built-in drivers
- Kernel space vs user space
- Kernel memory allocation
- Synchronization
- DMA
- Interrupt handlers
- Resource (I/O space) request
- Hardware access functions (read/write)
- Proc and sysfs filesystems
- Debugging
- Driver types
- Netfilter architecture

# Common Driver Interface

- init and exit
  - declared with module_init and module_exit macros
    - called at system initialization/shutdown time
    - for loadable modules, called when module inserted or removed from kernel
  - register/unregister device_driver struct:

```
struct device_driver vx1xx_driver = {
    .name       = "vx1xx-uart",
    .bus        = &platform_bus_type,
    .probe      = vx1xx_probe,
    .remove     = vx1xx_remove,
    .suspend    = vx1xx_suspend,
    .resume     = vx1xx_resume,
};
```

    - bus: used in device-driver matching (see next slide)
    - probe and remove
      - called when device registered/unregistered during system initialization
      - for “pluggable” devices, called when device “insertion” or “removal” detected
    - suspend and resume
      - called by power management subsystem to inform device to power down/up

# Device and Driver Model

- Goal: separate mechanism (driver) from config info (device)
- Device specification
  - Provided in platform-specific code (vx115_vep.c)
- Device hierarchy (parents)
- Registration
  - platform_add_devices, amba_device_register in board setup function
  - driver_register in driver init function; bus type in device_driver struct
- Device and driver matching and configuration
  - registering devices and drivers causes match to occur
    - driver's probe function called to configure driver with handle to device data
  - often text-based match (ex. platform devices)

# Loadable vs Built-in Drivers

- Virtually all drivers support both modes
- module_init macro
  - built-in driver: places function pointer in init section so will be called during system initialization
  - loadable module: just aliases function to init_module; called by module loader
- __init function qualifier: places function in init section so memory can be reclaimed after boot
- lsmod, insmod, rmmod and modprobe applications
  - lsmod: list currently loaded modules
  - insmod: loads specified module (need complete path)
  - modprobe: loads specified module and all modules it depends on
    - looks in /lib/modules for named module
    - uses modules.dep file generated by depmod to resolve module dependencies
    - if you add a new module, need to add new modules.dep to use modprobe

# Kernel Space vs User Space

- Each user application uses same virtual address space (usually 0-based)
  - MMU maps each app's virtual addresses to its personal physical pages; map changes on context switch
  - if give kernel pointer to userspace buffer and get context switch, what happens to buffer reference? :-(
- copy_from_user, copy_to_user
  - transfer between user process buffer and kernel buffer
  - make sure pages aren't swapped out (not an issue in most embedded systems)
- mmap and remap_pfn_range
  - map a kernel buffer so it can be directly accessed from user application
  - mmap function provided as part of driver interface (see below)
  - kernel function remap_pfn_range does actual mapping

# Kernel Memory Allocation

- kmalloc, kfree: allocate and free memory in kernel space
  - allocates virtually and physically contiguous buffer, returns virtual address
  - flag specifies whether can sleep or not during allocation

- vmalloc
  - allocates virtually contiguous buffer , returns virtual addresses
  - can allocate larger buffers, but less efficient

# Lists

- Use built-in Linux list functions
  - gives doubly-linked list
  - struct list_head
  - list_add, list_add_tail, list_splice, list_del, list_empty
  - list_entry(entry, type, member);

- container_of macro
  - get structure containing specified field
    - specified field need not be first field
  - container_of(ptr, type, member)
  - list_entry just #defined to container_of

# Synchronization: Semaphores and Spinlocks

- semaphores and mutexes
  - usual semaphore semantics
  - down_interruptible, down_trylock, up
  - applicable to thread context only (suspends)

- spinlocks
  - just disable/reenable interrupts in uniprocessor (non-SMP) system
  - spin_lock_irqsave, spin_unlock_irqrestore
  - protects against threads and ISRs

# Synchronization: Completions

- wait until signalled that some operation is complete
- use completion struct and functions
  - struct completion c;
  - init_completion(&c);
  - wait_for_completion(&c);  // wait until completion signalled
  - complete(&c); // to wake up a process waiting for completion
- wait applicable in thread context only (suspends); completion signalled from thread or ISR context

# Synchronization: Wait Queues

- sleep until awakened and specified condition true
- wait_event_interruptible(wait_queue, condition);
  - wait on queue until awakened and condition true
- wake_up_interruptible(wait_queue);
  - awaken waiting task(s)
- wait applicable in thread context only (suspends); wake_up can be signalled from thread or ISR context

- Notes
  - oddly, can't assume condition true when awakened
    - might be awakened due to signal
    - might have been out-raced by another task
  - should protect condition test with semaphore/spinlock
    - guard against race conditions

# DMA

- buffer allocation
  - dma_map_single/dma_unmap_single with kmalloc/kfree
    - kmalloc/kfree handle allocation
    - map functions handle cache coherency
      - transfer ownership of buffer to/from DMA controller
      - extra direction argument makes cache sync more efficient
    - also have dma_map_sg, dma_unmap_sg for mapping scatter-gather lists
  - dma_alloc_coherent, dma_free_coherent
  - allocates non-cacheable buffer; less efficient
- kernel DMA interface
  - request_dma, free_dma: request/free a DMA channel
  - set_dma_addr, set_dma_count, set_dma_mode, set_dma_sg: configure DMA channel
    - Note that set only single address; assumes DMA “target” dedicated for each channel
  - enable_dma, disable_dma: start or end DMA transfer

# Interrupt Handling

- Interrupt registration (low-level ISR)
  - request_irq(unsigned int irq, irq_handler_t handler, unsigned long flags, char *name, void *context);
- Interrupt handler (low-level)
  - irqreturn_t irq_handler(int irq, void *context, struct pt_regs *regs);
  - return IRQ_HANDLED, IRQ_NONE (not handled)
- Synchronization
  - Use spinlocks to protect against low-level IRQ handler
- “Bottom halves”
  - Defer interrupt processing - “high-level” interrupt handlers
  - Use tasklets and work queues to carry out processing

# Interrupt Handling: Tasklets

- Context
  - run in interrupt context (with interrupts enabled), so can't suspend
  - done as a softirq: run after all hardware interrupts processed
    - kernel calls do_softirq at end of low-level interrupt processing
  - runs once when scheduled

- Use
  - struct tasklet_struct tasklet;
  - void tasklet_handler(unsigned long data);
  - tasklet_init(&tasklet, tasklet_handler, data);
  - tasklet_schedule(&tasklet);    // schedule handler to be executed

- Synchronization
  - Use spinlocks to protect against tasklet

# Interrupt Handling: Work Queues

- Context
  - run in process context, so can suspend
  - run as kernel thread, so higher priority than user threads
  - runs once when scheduled

- Use
  - struct work_struct work;
  - void work_handler(void *context);
  - INIT_WORK(&work, work_handler, context);
  - schedule_work(&work);
  - Note: changed in 2.6.20; context replaced with pointer to work struct...

- Synchronization
  - Use semaphore to protect against work queue

# Resource Requests

- Request access to hardware region (registers, etc.)
  - request_region, release_region: I/O space request
  - request_mem_region, release_mem_region: memory region requests

# Resource Requests and Hardware Access

- Resource (I/O space) request
  - Request access to hardware region (registers, etc.) during driver initialization
  - request_region, release_region: I/O space request
  - request_mem_region, release_mem_region: memory region requests

- Hardware access functions (read/write)
  - readb, readw, readl, writeb, writew, writel
  - read/write 8/16/32-bit quantity from specified (virtual) address
  - Preferable for memory access over direct pointer references
    - intends to make drivers portable to systems with separate I/O space
    - Less relevant with embedded system

# Proc Filesystem

- “Virtual” directory created and maintained by kernel
  - appear as entries under /proc
- Provides control and statistics interface from userspace into drivers
  - just read like would with normal files (can use cat, e.g.)
  - Functions implemented by drivers which wish to expose an interface
- Use
  - #include <linux/proc_fs.h>
  - create_proc_entry, remove_proc_entry
    - request kernel to create entry for driver (usually during driver init)
    - specify parent directory within /proc, functions for read and write
  - proc read function
    - just return info about driver (often text)
  - proc write function
    - use supplied info to control the driver

# Debugging

- JTAG
  - best for kernel code and built-in drivers
  - not so useful for loadable modules or app code
- printk
  - Usual method of kernel and driver debugging: print messages to system log and console
- procfs
  - Can read out driver statistics/state
- objdump
  - Inspecting binaries (symbol info, disassembly, etc.)
- ksymoops
  - Decode kernel stack dumps into readable messages

# Driver Types

- Driver interface depends on type
- Character
  - stream- or character-oriented devices (UARTS, GPIOs,
- Block
  - Block-oriented devices (disks, etc.)
- Network
  - Drivers for network devices (Ethernet, Wifi, etc.)
- Higher-level frameworks
  - Driver provides interface required by higher-level framework
  - USB
  - MTD
  - SD/MMC
  - ...

# Character Drivers

- Interface: file operations (fops) struct

```
struct file_operations {
  – struct module *owner;
  – int (*open) (struct inode *, struct file *);
  – int (*release) (struct inode *, struct file *);
  – ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
  – ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
  – int (*mmap) (struct file *, struct vm_area_struct *);
  – loff_t (*llseek) (struct file *, loff_t, int);
  – unsigned int (*poll) (struct file *, struct poll_table_struct *);
  – int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
  – int (*flush) (struct file *);
  – int (*fsync) (struct file *, struct dentry *, int datasync);
  – ... (many more fields – note order changed!)
};
```

- Implement functions for open, close, read, write, seek, etc.
  - Can leave many null if don't care about operation

# Character Driver Registration

- major/minor number reservation
  - major/minor used to map /dev node to driver
  - register_chrdev_region(dev_t from, unsigned count, const char *name)
    - reserve range of major/minor numbers for device

- driver registration
  - have driver data structure
    - contains fields for whatever driver needs to do its work (buffers, lists, ...)
    - has embedded cdev struct
  - use cdev_init(cdev, fops) to initialize embedded cdev struct with file ops
  - use cdev_add(cdev, device_number, range) to register embedded cdev struct with kernel

# Driver Struct, Inodes and Files

- Issue: how to get driver data structure for use in fops functions
  - can use static struct, but limits number of devices supported by driver
  - better: allocate struct for each device, stash so passed in as function arg

- Approach: stash device struct pointer in file structure
  - open function passes inode and file pointers
    - inode has pointer to driver's cdev
      - was initialized when cdev_init called
  - extract pointer to driver struct which contains cdev struct with container_of
  - set file->private_data field so can retrieve driver struct when get other calls
    - other file_operations functions pass just file struct, not inode

# Activity and Resources

- Activity
  - use objdump and ksymoops
  - Create simple character driver
    - File ops interface
    - Proc interface
    - See example driver source code
      - Can build on x86 platform; has “device” module that registers device that matches with driver

- Resources
  - Linux Device Drivers, 3rd edition, Alessandro Rubini, O'Reilly
    - online version at http://lwn.net/Kernel/LDD3/ (pdf)
    - 2nd edition in HTML: http://www.xml.com/ldd/chapter/book/
  - Linux Kernel Development, Robert Love, Sams

# Embedded Linux Application Development Overview

# Embedded Linux Application Development

- C development
- Posix development
- Makefiles
- Driver interface
- Library linking
- Debugging
- C++
- Shell script development

- Activities
- Resources

# C Application Development

- Just regular Unix Posix development
- processes: fork, exec, wait
- threads: pthreads and attributes
- synchronization: condition variables, semaphores and mutexes
- communication: pipes, queues, shared memory
- file I/O: open, close, read, write
- signals
- sockets for networking

# C Application Development (cont.)

- Driver interface
  - file nodes and standard file operations
- Libraries
  - Toolchain provides standard C libs
  - Specify paths to custom libs and headers
- C++
  - standard C++ development
  - Use libstdc++

# Debugging

- Debugging with printf
  - Send messages to console or system log

- Debugging with gdbserver
  - Build gdbserver for platform
  - Build app with debugging symbols (-g when compiling)
  - Start app to be debugged with gdbserver
    - gdbserver <serial-device> <app-to-debug>
    - gdbserver /dev/ttyS1 /bin/ls
  - Connect to gdbserver over serial with gdb-capable debugger
    - gdb, Insight, etc.

# Shell Script Development

- Use standard apps in shell script
- Pipes, redirection
- if, case
- Environment variables
- Notes
  - Different shell variants have different syntax
  - Arithmetic a pain

# Activity and Resources

- Activity
  - Debug an app with gdbserver

- Resources
  - POSIX specs:  http://www.unix.org/single_unix_specification/
  - Advanced Programming in the UNIX Environment, Richard Stevens, Addison-Wesley, 1992, http://www.kohala.com/start/

# Open-Source Software Licenses

# Open-Source Software Licenses

- GPLv2
- Common properties
- LGPL
- MIT, modified-FreeBSD

- Resources

# Common Properties

- Use at your own risk
  - no guarantee
  - don't sue me if it doesn't work

- Issues
  - unknowingly incorporating software which contains patented material
  - combining software with incompatible licenses

# GPLv2

- GNU Public License
- Must deliver source together with binary to customers
  - no customer, no delivery (internal corporate uses)
  - no requirement to “feed back” mods or make them “publicly” available – just must make source available to “customer” if deliver software
- Examples
  - Linux kernel
- Pros
  - tend to get mods fed back to common software baseline – everybody benefits
- Cons
  - linking extends GPL to non-GPL software - must provide source for all software linked with GPL software

# LGPL

- Lesser or “Library” GPL
  - Software linked with LGPL software not covered by LGPL - source delivery not required
  - Source code of LGPL code itself (together with any mods) must be made available to customer
- Examples
  - glibc
- Pros
  - Has allowed for non-open-source Linux application development
    - Situation less clear for kernel code such as loadable modules
- Cons
  - Still required to deliver source of libraries

# MIT, Modified-BSD

- No source delivery required
- Pros
  - preferred by businesses worried about exposing proprietary stuff
- Cons
  - has led to fragmentation (e.g., multiple BSD implementations)
  - slower progress (e.g., no good open-source Flash filesystem implementation in BSD's, even though iPhone uses BSD-derived OS)

# Resources

- Open-source software licenses described: http://www.gnu.org/licenses/license-list.html
- Understanding Open Source and Free Software Licensing, Andrew St. Laurent, O'Reilly, 2004, http://www.oreilly.com/catalog/osfreesoft/book/

# Linux Tools and Resources

# Tools

- gcc cross-compilation toolchain
  - Pre-built: Code Sourcery: http://www.codesourcery.com/
  - Build your own: Dan Kegel's CrossTool: http://kegel.com/crosstool/
- Insight (includes gdbserver): http://sourceware.org/insight/
- Ksymoops

# Resources - Books

- Linux Device Drivers, 3rd edition, Alessandro Rubini, O'Reilly
  - online version at http://lwn.net/Kernel/LDD3/ (pdf)
  - 2nd edition in HTML: http://www.xml.com/ldd/chapter/book/
- Linux Kernel Development, Robert Love, Sams
- Building Embedded Linux Systems, Karim Yaghmour, O'Reilly
- Embedded Linux: Hardware, Software and Interfacing, Craig Hollabaugh, Addison Wesley
- Understanding Open Source and Free Software Licensing, Andrew St. Laurent, O'Reilly, 2004
- Advanced Programming in the UNIX Environment, Richard Stevens, Addison-Wesley
- Kernel Documentation subdirectory

# Resources - Web

- Linux kernel cross-reference website: http://lxr.linux.no/
- Linux Device Drivers, 3rd edition, Alessandro Rubini, O'Reilly
  - online version at http://lwn.net/Kernel/LDD3/ (pdf)
  - 2nd edition in HTML: http://www.xml.com/ldd/chapter/book/
- ARM Linux website: http://www.arm.linux.org.uk/
  - arm-linux-kernel mailing list
- CELF Wiki:  http://tree.celinuxforum.org/pubwiki/moin.cgi
- CELF Embedded Linux Conference: http://www.celinux.org/elc2007/index.html
- Linux Journal: http://www.linuxjournal.com/
- Linux Magazine: http://www.linux-mag.com/
- POSIX specs:  http://www.unix.org/single_unix_specification/

# Resources – Web (cont.)

- Porting the Linux Kernel to a New ARM Platform (2.4-series kernel): http://linux-7110.sourceforge.net/howtos/netbook_new/porting2arm_aleph.pdf
- Linux Porting Guide (uses MIPS as example): http://www.embedded.com/shared/printableArticle.jhtml?articleID=9900048
- Linux kernel source repository: http://www.kernel.org
- Busybox: http://www.busybox.net
- uClibc: http://www.uclibc.org/
- Qtopia: http://www.trolltech.com/products/qtopia/phone.html
- Open-source software licenses described: http://www.gnu.org/licenses/license-list.html

GICL