Migrating Legacy RTOS Device Drivers to Embedded Linux

Bill Weinberg, LinuxPundit.com
Webinar - December 9, 2008
• **Objectives**
  - Shed light on the path from Legacy to Linux
  - Analyze and critique both Legacy and Linux
  - Provide useful, actionable information to ease migration

• **Non-Objectives**
  - Teach attendees how to write Linux and/or Legacy drivers
  - Cover every possible type of legacy driver or RTOS
• Legacy Landscape

• Migrating Drivers and I/O Constructs to Linux

• Challenges and Choices

• Resources for Migration

• Return on Migration Investment

• Conclusion
Migration Decision Tree

Legacy Driver

Device Support
- Linux kernel tree
- Semiconductor suppliers
- SBC Vendors / IHVs

Linux Driver Available?

Driver Up-to-date?

Outsource?

Services Organization
- Linux Supplier (e.g., MV)
- Consultant
- Linux Drivers Project

In-House Migration
- Utilize internal resources
- Leverage outside help

Migrated Driver
Migration Decision Tree

Legacy Driver

Check for device support
• Linux kernel tree
• Semi suppliers
• SBC Vendors / IHVs

Linux Driver Available?

Driver Up-to-date?

Outsource?

N

Y

Services Organization
• Linux Supplier (e.g., MV)
• Consultants
• Linux Drivers Project

In-House Migration
• Utilize internal resources
• Leverage outside help

Main Webinar Focus

Migrated Driver
<table>
<thead>
<tr>
<th>Legacy Embedded OS Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMX</td>
</tr>
<tr>
<td>AvSys</td>
</tr>
<tr>
<td>C Executive</td>
</tr>
<tr>
<td>CMX</td>
</tr>
<tr>
<td>eCOS</td>
</tr>
<tr>
<td>Embedix</td>
</tr>
<tr>
<td>iRMX</td>
</tr>
<tr>
<td>iTRON / µiTRON</td>
</tr>
<tr>
<td>µcos</td>
</tr>
<tr>
<td>MQX</td>
</tr>
<tr>
<td>MTOS</td>
</tr>
<tr>
<td>Nucleus</td>
</tr>
<tr>
<td>OSE</td>
</tr>
<tr>
<td>OS-9</td>
</tr>
<tr>
<td>PDOS</td>
</tr>
<tr>
<td>pSOS</td>
</tr>
<tr>
<td>QuickTask</td>
</tr>
<tr>
<td>Raven</td>
</tr>
<tr>
<td>REAL/IX RTexec</td>
</tr>
<tr>
<td>REX</td>
</tr>
<tr>
<td>ROM-DOS</td>
</tr>
<tr>
<td>RT-11</td>
</tr>
<tr>
<td>RTXC</td>
</tr>
<tr>
<td>Skympx</td>
</tr>
<tr>
<td>SMX</td>
</tr>
<tr>
<td>SuperTask!</td>
</tr>
<tr>
<td>TargetOS</td>
</tr>
<tr>
<td>THEOS</td>
</tr>
<tr>
<td>Threadx</td>
</tr>
<tr>
<td>VRTX</td>
</tr>
<tr>
<td>VxWorks</td>
</tr>
<tr>
<td>. . .</td>
</tr>
</tbody>
</table>
Webinar Focus

Basis for this Webinar

Majority of Designs and Deployments

VxWorks (Wind Kernel)

“VRTX Works”

VxWorks 5.2

VxWorks 6.0
Classic Legacy RTOS Architecture

Applications

Java

Other Libs

Graphics

Enabling Middleware

Adaptation Layer

Network & FS

RTOS

H/W
Classic Legacy RTOS Failure

- Simple kernel cannot cope with complex s/w stack
- Poor or non-existent off-the-shelf s/w and m/w
- Single-vendor solution
- No support for deployed memory protection
- Mismatch between modern CPUs and 15-year old RTOS

- Cannot support large teams of s/w engineers
Where do Legacy RTOS Drivers Reside?

- Device interface code can be co-mingled with almost any layer in the legacy software stack

- Complicates
  - Re-use
  - Debugging
  - Maintenance
Challenges from Legacy Spaghetti I/O

• **Applications, libraries, drivers, kernels share code**
  - Ties “user code” and drivers to kernel
  - Confounds attempts at determining context
  - Makes migration more difficult

• **Requires global re-entrancy**
  - Code used by tasks, ISRs, etc.

• **Complicates coding, macros**
  - Preambles and codas
  - RTS or RTI?
### Key Attributes of Legacy OSes and Linux

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Legacy OS</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Model</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MMU-enabled</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>User Space Addressing</td>
<td>Physical</td>
<td>Virtual</td>
</tr>
<tr>
<td>Kernel Space Addressing</td>
<td>Physical</td>
<td>Virtual</td>
</tr>
<tr>
<td>Driver Model</td>
<td>None/Partial</td>
<td>Complete</td>
</tr>
<tr>
<td>Interrupt Service</td>
<td>Anywhere</td>
<td>Driver Only</td>
</tr>
</tbody>
</table>
Legacy APIs and Constructs

• **Legacy OSes boast 100s of calls**
  • VxWorks has 1200+ “subroutines”
    • Mix of user and system functions
  • Organized into approximately 50 libraries

• **Key VxWorks Libraries**
  • Kernel - kernelLib
  • General I/O
    • ioLib, iosLib
    • intLib - interrupt subroutines
  • Asynchronous/Serial I/O
    • aioPxLib, aioSysDrv, ptyDrv, tyLib, tyCoDrv
  • Networking
    • IfLib and if_* drivers
    • netLib, netDrv
  • Disk - ideDrv, ramDrv, wd33c93Lib
  • Memory
    • memDrv, memLib, memPartLib
  • Timers
    • tickLib, timerLib
• **Formal Appearance**
  
  • Like UNIX/Linux, VxWorks offers Formal Driver Entry Points
  
  • Seven key calls implemented in block and character drivers
    • creat( ), remove( ), open( ), close( ), read( ), write( ), and ioctl( )
    • Plug into Wind BSP model
    • Maps most cleanly to Linux

  • Wind River and partners offer(ed) formal h/w interfaces, a.k.a. *device drivers*
    • Supplied in binary-only distribution
    • Bundled with h/w or in kit form
• **Ad Hoc Reality**
  - OEMs / End-users of VxWorks and other RTOS crafted ad hoc device interfaces
    - Ported from other legacy systems - e.g., pSOS
  - Often replaced VxWorks/RTOS subsystems wholesale
    - E.g. TCP/IP stack and network drivers

• **Proprietary I/O Subsystems**
  - Sui generis APIs
  - In-line memory-mapped I/O
  - RYO memory/buffer management
  - Library code shared with
#define DATA_REGISTER 0xF00000F5

char getchar(void) {
    return (*((char *) DATA_REGISTER));
}

void putchar(char c) {
    *((char *) DATA_REGISTER) = c;
}
Classic RTOS I/O - Producer-Consumer

- All code in system mode
  - Nominally faster
  - JSR instead of TRAP
  - Corruptible
- I/O code spread across whole stack
- Blurring between application and system contexts
- Tasks must perform own synchronization
Compare to Linux Driver Model

- Drivers link or insert into Linux kernel
  - Static drivers
  - Modules
  - Special user-space I/O
- I/O code segregated to kernel and libraries
- Strict separation between application and system contexts
- Kernel synchronizes calls based on thread priority
Agenda

- Legacy Landscape
- Migrating Drivers and I/O Constructs to Linux
- Challenges and Choices
- Resources for Migration
- Return on Migration Investment
- Conclusion
Driver Migration
Legacy RTOS Architecture

Interrupt

ISR  Driver  Scheduler  Task

Single Physical Address Space

- **Interrupt Context**
  - In theory, first part of driver
  - In practice, ISR/interrupt_disable could include entire driver

- **Deferred Execution**
  - Prioritized “normal” task
  - Runs in system context, like everything
Deferred Execution Mechanisms - A surfeit of choices

- SoftIRQ: prioritized but restricted
- Bottom Half: legacy, FIFO
- Tasklet: schedulable kernel entity (not a kernel thread)
- User Space I/O: schedulable user space entity
VxWorks Network Architecture

- VxWorks Network MUX and Drivers

![Diagram showing the network architecture with MUX at the center, connected to IP and ICMP, Other Protocols, Ethernet, Wireless, Backplane, and Other.]
- **Two ways to access Linux kernel APIs**
  - Static Linkage - ALL kernel symbols available
  - Dynamic Linkage - Available to modules
    - Symbols exported with kernel macro

- **APIs and Symbols in /proc/kallsyms**
  - Small appliance built on 2.6.19: 28,004 symbols
  - FC 8 desktop built on 2.6.23: 41,420 symbols

- **Which APIs to use?**
  - Depends on your type of driver
    - Resource allocation: kmalloc(), kfree(), etc.
    - Bus parsing: pci_bus*, etc.
    - Interrupt and Preemption: cli(),
    - Scheduling: schedule()
Drivers (like the kernel) run in a virtual address space

- Physical locations must be mapped into logical addresses

Example

- Employs user-space APIs - try it yourself!

```c
#include <sys/mman.h>

#define REG_SIZE   0x4   /* device register size */
#define REG_OFFSET 0xFA400000
    /* physical address of device */

void *mem_ptr;
    /* de-reference for memory-mapped access */
int fd;

fd=open("/dev/mem",O_RDWR);
    /* open physical memory (must be root) */

mem_ptr = mmap((void *)0x0,
    REG_AREA_SIZE, PROT_READ+PROT_WRITE,
    MAP_SHARED, fd, REG_OFFSET);
    /* actual call to mmap() */
```
Agenda

• Legacy Landscape

• Migrating Drivers and I/O Constructs to Linux

• Challenges and Choices

• Resources for Migration

• Return on Migration Investment

• Conclusion
Choices and Challenges
The Mutable Linux Kernel API Set

• **From version 1.0 (or before) Linus pledged**
  • Users and user programs could depend on API stability
    • Maintained good continuity in libraries and system calls
  • Kernel developers would be free to innovate

• **Innovation vs. Stability**
  • Kernel/Driver APIs can change across kernel releases
  • Forces drivers to be specific to kernel version
    • Greater issue for (binary) modules than for kernel drivers
  • Can induce need for back and forward porting

• **Creates tension between kernel developers and OEMs**
  • Kernel community thinks OEMs are “hoarding”
  • OEMs and others want “future proof” drivers
• **Best Practices**
  • Only use public symbols (EXPORT_SYMBOL)
    • In both modules AND kernel drivers
  • Work with community / ecosystem to mainline driver code
    • Bring driver code into kernel sources and/or LDDP
    • Work with kernel developers, h/w vendors, et al.
  • Build in user-space as much as possible
    • Much legacy driver code better implemented as daemons
    • Leverage usblib
    • Run deferred I/O processing as user-space programs
• **Community Work**
  • Lobby kernel developers, Linux Foundation to stabilize APIs
  • Invest in community user-space I/O projects
**Challenges and Choices**

**Modules vs. Kernel Drivers**

- **Key Question - which to use?**
  - Attributes and Trade-offs

<table>
<thead>
<tr>
<th></th>
<th>Kernel Drivers</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Image Impact</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Available at Boot</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Available APIs</td>
<td>All</td>
<td>Many/Most</td>
</tr>
<tr>
<td>License</td>
<td>GPL</td>
<td>???</td>
</tr>
</tbody>
</table>
**Choices and Challenges**

**Licensing and IP**

- **Kernel Drivers**
  - Built as part of the Linux kernel: must be GPLv2

- **Modules**
  - Ambiguous status
    - Not protected by syscall interface exception in COPYING
    - Linus: “There is NOTHING in the kernel license that allows modules to be Non-GPL”
    - However, there is nothing in the COPYING that requires modules to be GPL, either
  - Community voices adamant
    - GKH: “Binary modules are illegal”

- **Best Practices**
  - License drivers/modules as GPL whenever possible
  - Ship binary modules only if
    - You have clear, pre-existing copyright
    - You can show independent derivation of module code
Linux Runtime Architecture and Licensing

**Key**
- GPL
- LGPL
- Any License

**“User Space”**
- Application
  - GPL Libraries
  - GPL Code
  - LGPL Libraries
  - Other Libraries

**“Kernel Space”**
- Linux Kernel
- Boot-time Device Drivers

System Call Interface
Choices and Challenges
Linux System Debugging - Virtual Addressing

• **Boot vs. Run-time**
  • Initial boot phase occurs with physical addresses
  • After MMU enabled, ALL addressing is virtual

• **Physical vs. Logical addressing**
  • Page mapping and swapping
  • Breakpoint offsets
  • ICE and JTAG support
  • Good news - kernel allocated as contiguous RAM

• **Shared Libraries**
  • Present special challenges
  • Can have different addresses for different processes
Debugging - “Kernel-awareness”

• Legacy RTOS debuggers boasted “Kernel “Awareness”
  • e.g., pRISM+ for pSOS+, some features/versions of Tornado
  • Debugger “understood”, elucidated kernel constructs
    • Tasks, locks, memory buffers, other data structures

• Linux kernel debuggers are non kernel-aware
  • At least not in the sense of RTOS debuggers
  • Treat Linux kernel source code like any application

• Analysis and display of system attributes
  • /proc file system
  • LTT
  • Commercial Tools
    • E.g., Viosoft Arriba, MontaVista devRocket
Agenda

• Legacy Landscape

• Migrating Drivers and I/O Constructs to Linux

• Challenges and Choices

• Resources for Migration

• Return on Migration Investment

• Conclusion
Migration Resources
Commercial Tools and Technologies

• **API Toolkits**
  • Translate APIs among RTOSes, Linux (e.g., MapuWare Oschanger)

• **Driver Abstraction Tools**
  • Environments for building OS-independent drivers and re-using legacy code
    • MapuWare WinDriver

• **Virtualization**
  • User virtual machines to
    • Host entire RTOS stack including drivers
    • Encapsulate legacy drivers in “lightweight execution environments”
  • Embedded hypervisors / VM platforms
    • Open Kernel Labs
    • Virtual Logix
    • VMware
Migration Resources
Open Source Projects and Technologies

- **V2Linux.org**
  - Freshly updated site sponsored by MontaVista

- **V2Lin**
  - Update to original V2Linux/p2Linux projects (circa 2001)

- **NDISwrapper**
  - Supports encapsulation, reuse of Windows network drivers
  - Primarily useful for x86 PCI/USB NICs

- **User-Space Device Drivers**
  - User Level Device Drivers for Linux - Gelato
    - [http://www.gelato.unsw.edu.au/IA64wiki/UserLevelDrivers](http://www.gelato.unsw.edu.au/IA64wiki/UserLevelDrivers)
  - USB - libUSB is now mainstream
• **Linux Driver Project**
  • Goal to create FOSS drivers for broad range of h/w
  • Sponsored by the Linux Foundation
  • Three outputs
    • Mainline device drivers
    • Out-of-tree drivers
    • Devices needed
  • Led by Linux kernel developer Greg Kroah Hartman
    • Current roster of 200+ developers
• **Definitive text on Linux Device Drivers**

• **Most Current Version**
  • Third Edition - 2.6 kernel
  • Includes info on new constructs
  • Greg KH joins Corbet and Rubini
• **Many options**
  - Linux distribution and tool kit suppliers
  - Integrators and service providers
  - Individual consultants

• **Caveats**
  - Migrating drivers is not generic consulting!
  - PS supplier must have expertise in
    - Your Linux platform (commercial or RYO)
    - Legacy OS architecture
    - Device technology
    - Writing device drivers!
• **Legacy Landscape**

• **Migrating Drivers and I/O Constructs to Linux**

• **Challenges and Choices**

• **Resources for Migration**

• **Return on Migration Investment**

• **Conclusion**
• **Migration to Linux**
  - Benefits well-known
  - Greater control, lower costs, more scalability, vendor independence, modern OS, etc.

• **Migration from Legacy**
  - Preserve legacy investment as much as possible
  - Freedom from single supplier and ecosystem

• **Migration to Linux Mainstream**
  - Share support burden
  - Ease migration forward with Linux kernel
• Legacy Landscape

• Migrating Drivers and I/O Constructs to Linux

• Challenges and Choices

• Resources for Migration

• Return on Migration Investment

• Conclusion
Conclusion

• **Driver Migration Involves Real Investment**
  • Usually entails multiple devices / interfaces
  • Need to capture legacy code, technology, knowledge

• **Presents Your Organization with**
  • Challenges
    • Changes in design, practices, scope of code
    • May need to deprecate s/w and h/w architectures
    • Need to (re)train existing team, add new expertise
  • Opportunities
    • Optimize platform, improve performance
    • Unify fragmented internal platforms, code bases
    • Create a more maintainable foundation for future
    • Join with mainstream in embedded and enterprise