Building embedded and dedicated applications using Linux and other O/Ss

Bill Rousseau, Software Consultant
w.rousseau@ieee.org
http://www.ieee-sv-consult.org

© 2003 by William F. Rousseau
Scope

- New concerns for applications programmers
- Possible O/Ss
- Real-time O/Ss
- Simple Linux application experience
- The GPL license
- Home brew O/S
- Building Linux kernel
Applications
programmers face new system issues

• Potential race conditions (hard-soft)
• Preemption for fast response
• Potential race conditions (soft-soft)
• Locks, atomic operations, and ring buffers to avoid races
• Deadlocks
• Need for volatile keyword
• Priority scheduling
• Priority inversion
• Deadline task scheduling
  – Needed?
  – Desirable
  – Undesirable
What are possible O/Ss?

- DRDOS
- SCO Unix/Unixware
- VxWorks
- QNX
- Windows/Windows CE
- eCos
- RT Linux/Preemptive Linux/Linux
- Home made
Linux offers significant advantages:

- No license fees—important for 1000s of systems
- All the normal Unix features and utilities
- Small footprint possible
- Robust
RT can mean two things

• Direct digital control/time critical/hard RT
  – A-D samples, D-A settings
  – Binary signals and settings

• Supervisory control/less time critical/soft RT
  – Buffered by hardware
  – Adjust set points
  – GUI
Some O/Ss are suited to one and some to both

• Direct digital—RT Linux, eCos, QNX, VxWorks, home made, Windows CE

• Supervisory—RT Linux, Preemptive Linux, SCO, Unixware, Windows CE/other Windows
RT Linux is two O/Ss

RT O/S

RT tasks
Linux
Linuxes have different kernel latency:

- Typical nos. for ~700 MHz Pentium:
  - RT-Linux <3 ms
  - Preemptive Linux: ~100 ms (average much less)

Typical worst case results:
- Linux should be ~2 µsecs.
- RT-Linux should be ~10 µsecs.
Kernel latency does make a difference more often than one might think

- Solaris 2.7 (preemptive) ~4000 credit card transactions/minute
- Linux 2.2 (non-preemptive) ~3800 credit card transactions/minute
- Average transaction time slightly less on Linux
- Outliers made the difference
Interrupt and context switch latency are also important

- Real interrupt latency is affected by interrupt handlers—most O/Ss are in the 5-100 μ sec range
- Linux is relatively fast for process
- Preemption may require stack switch
<table>
<thead>
<tr>
<th>O/S Capabilities</th>
<th>General</th>
<th>Embedded</th>
<th>Dedicated</th>
</tr>
</thead>
<tbody>
<tr>
<td>File system</td>
<td>Yes</td>
<td>Windows</td>
<td>Yes</td>
</tr>
<tr>
<td>Cooperative MT</td>
<td>Yes</td>
<td>Sometimes</td>
<td>Yes</td>
</tr>
<tr>
<td>Preemptive MT</td>
<td>Yes</td>
<td>Yes</td>
<td>Maybe</td>
</tr>
<tr>
<td>Interrupts</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sophisticated scheduling</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Program loading</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Shared libraries</td>
<td>Yes</td>
<td>Usually</td>
<td>Yes</td>
</tr>
<tr>
<td>Job execution</td>
<td>Yes</td>
<td>Seldom</td>
<td>Usually</td>
</tr>
<tr>
<td>Resource protection</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Inter-task com.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>O/S Capability</td>
<td>General</td>
<td>Embedded</td>
<td>Dedicated</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>Atomic ops. and locks</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Peripheral support</td>
<td>Yes</td>
<td>Seldom</td>
<td>Sometimes</td>
</tr>
<tr>
<td>Utilities</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GUI</td>
<td>Yes</td>
<td>Yes</td>
<td>Seldom</td>
</tr>
<tr>
<td>Network cap.</td>
<td>Yes</td>
<td>Sometimes</td>
<td>Yes</td>
</tr>
<tr>
<td>Portability</td>
<td>Yes</td>
<td>Seldom</td>
<td>Sometimes</td>
</tr>
</tbody>
</table>
Credit Cards over Internet Pre-prototype schedule:

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select/</td>
<td>Modify</td>
<td>Make 2-diskette Linux</td>
<td>Test and</td>
<td>Conting</td>
<td>Hardware</td>
<td>AM</td>
</tr>
<tr>
<td>order</td>
<td>test bed software</td>
<td>system</td>
<td>debug on</td>
<td>ency</td>
<td>arrives</td>
<td>on DoC</td>
</tr>
<tr>
<td>hardware</td>
<td></td>
<td></td>
<td>PC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Install on DoC</td>
<td>Test and debug</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arrives</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contingency</td>
<td>BoD meeting AM</td>
</tr>
</tbody>
</table>
PC/104 processor board
VersaLogic Corporation Bobcat 586 Processor
Project didn't go as planned

• First week was ok
• Hardware arrived late PM Tuesday
• Hardware was flacky
  – Didn't work reliably until powered up for 2 hours
  – Disk-On-a-Chip didn't work
• Distributor unhelpful
Solution:

- Power up the night before meeting
- Make a 1-diskette bootable floppy version
- Use ramdisk for root filesystem

A whole working Linux system and a simple application fit on one diskette!
Mini-distribution built

- Startup was simple
- Mounts
- Used ramdisk and RO file system
- Slackware
- Utilities, libraries, etc. came from Slackware
- Installation/repair
- Boot patterned after 2-diskette from Slackware

Mini-distribution built
Linux (or Unix) startup need not be complex

- Kernel is loaded
- Root filesystem is mounted
- Kernel starts init; init reads inittab
- Init runs startup sh scripts (as root)
- Scripts start daemons and does housekeeping
- Complexity is needed for multiuser servers
Init also performs other management tasks

- Manages login-related programs
- Starts run-level change scripts
- Discards exit codes from orphans
- Starts shutdown and reboot scripts
Simple rules protect proprietary code from GPL

• Use shared libraries
• Make proprietary kernel extensions loadable modules
• Using gcc and header files is usually ok*

* Defines and macros in header files must be less than a quite large maximum size
Changes to GPL code require you to:

- Note your change in source
- Make source available for three years
  - Free (e.g. on web site)
  - With product
  - Media charge for copy

Avoid patents or other restrictions.
Exercise due diligence

• Read the GPL and LGPL
• Make sure what license applies to code or tools you use
• If warranted, obtain legal advice from a qualified attorney
You want to build your own O/S:

Minimal O/S is a scheduler

• For really simple needs a loop design may work
• Alternatively a loop design with interrupt handlers
• More complicated:
  – A cooperative MT scheduler
  – A preemptive MT scheduler
Multi-tasking can be cooperative or preemptive

<table>
<thead>
<tr>
<th>Cooperative</th>
<th>Preemptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>• simple</td>
<td>• more responsive</td>
</tr>
<tr>
<td>• avoid race and lock problems</td>
<td>• protection from errant tasks</td>
</tr>
<tr>
<td>• ugly SMs</td>
<td>• complicated</td>
</tr>
<tr>
<td>• errant task can be a problem</td>
<td>• race and lock problems</td>
</tr>
</tbody>
</table>
Home brew O/S was powerful enough for spacecraft

Requirements (partial):
• RISC cpu selected by Mgt.
• Precise, responsive attitude control
  – Rate gyros
  – Startracker–processing delay
  – DDC of thrusters, measurements
• Many other tasks
Key part was a Kalman filter

- Kalman filter is a model of the system with a feedback loop
- Measurements delays can be corrected
- Control actions can be projected forward with good accuracy
Successful design used the cooperative multitasking model

- One timer interrupt
  - Preplanned thruster schedule and measurements and poll otherwise
- Priority tasks
  - Kalman filter computation
- Background tasks
  - Startracker image processing
Linux is ported to many architectures:

- i386 (etc.) and Intel 64 (Itanium)
- Alpha (DEC/Compaq/HP-Compaq)
- Mips and Mips 64
- PArisk (HP)
- S390 (IBM mainframe)
- Sparc and Sparc 64 (Sun)
- PowerPC (IBM and Apple)
- ARM
Cross builds are quite possible

- This is complicated–avoid it if you can
- Try to debug as much as possible on a Linux workstation
- See references
Setting up Linux for a kernel build

• Get Linux: www.kernel.org
• Uncompress and untar in /usr/src
• mv linux linux-<version>
• ln -s linux-<version> linux
• cd linux
Use make menuconfig

or make xconfig
Now build it

build computer

Be sure you don’t clobber setup on

• make dep
• make bzImage
• make modules
• make modules_install
• cp arch/<arch>/boot/bzImage <dest>
What if I get an error?

- `make bzImage >log 2>&1`
- Identify source of error in log
- Possible causes
  - One selected option requires another
  - Patched kernel does not support option
  - Driver upgrade missed in kernel version
- Use `grep` on `.c`, `.h`, and Makefiles
One can make kernel changes

- SGI added its IRIX SCSI subsystem to Linux
- Linux header file parameter changes are fairly safe
- Modify device drivers
- Write your own device drivers
Conclusions

• Linux applications aren’t difficult
• The GPL license need not contaminate proprietary code
• Home brew O/Ss are feasible
• Building a custom Linux kernel is not difficult
References
Hollabaugh, Craig, Embedded Linux: Hardware, Software, and Interfacing
www.embeddedlinuxinterfacing.com
[Strong on cross build; see website for script.]

Embedded Linux Consortium
www.embedded-linux.org
[Proposing a standard.]
MontaVista Software
www.mvista.com
[Preemptive kernel.]

Linux Journal
www.linuxjournal.com

Embedded Systems Programming
www.embedded.com
Silicon Penguin
www.siliconpenguin.com

FSMLabs
www.fsmlabs.com
[RT Linux.]

Just do a web search