RTLinux
and embedded programming

Victor Yodaiken

Finite State Machine Labs (FSM)

Outline.

- The usual: definitions of realtime.

Outline.

- The usual: definitions of realtime.
- Who needs realtime?

Outline.

- The usual: definitions of realtime.
- Who needs realtime?
- How RTLinux works.

Outline.

- The usual: definitions of realtime.
- Who needs realtime?
- How RTLinux works.
- Why RTLinux works that way.

Realtime versus Time Shared

- *Time sharing software*: switch between different tasks fast enough to create the illusion that all are going forward at once.

Realtime versus Time Shared

- *Time sharing software*: switch between different tasks fast enough to create the illusion that all are going forward at once.
- *Realtime software*: switch between different tasks in time to meet deadlines.

Hard realtime

1. *Predictable performance* at each moment in time: not as an average.
2. *Low latency* response to events.
3. *Precise scheduling* of periodic tasks.

Soft realtime

- *Good average case performance*
- *Low deviation from average case performance*

Traditional problems with soft realtime

- The chips are *usually* placed on the solder dots.

Traditional problems with soft realtime

- The chips are *usually* placed on the solder dots.
- The machine tool *generally* stops the cut as specified.

Traditional problems with soft realtime

- The chips are *usually* placed on the solder dots.
- The machine tool *generally* stops the cut as specified.
- The power *almost always* shuts off before the turbine explodes.

New problems with soft realtime

- The voice-over-IP system only loses packets under stress.

New problems with soft realtime

- The voice-over-IP system only loses packets under stress.
- The cell-phone connect won’t drop your Internet handset during a handoff unless there is heavy traffic.

New problems with soft realtime

- The voice-over-IP system only loses packets under stress.
- The cell-phone connect won't drop your Internet handset during a handoff unless there is heavy traffic.
- If you start a browser on one machine, *most of the time* sales transactions won't get lost on the other side of your SOHO router.

You don’t really need hard realtime

- We have one failure every 100 hours of operation on the average, but some customers experience 5 failures in an hour.

You don’t really need hard realtime

- We have one failure every 100 hours of operation on the average, but some customers experience 5 failures in an hour.
- Tell ’em to stop being such whiners.

You don’t really need hard realtime

- We have one failure every 100 hours of operation on the average, but some customers experience 5 failures in an hour.
- Tell ’em to stop being such whiners.
- Poor performance? More memory and a faster processor!

You don’t really need hard realtime

- We have one failure every 100 hours of operation on the average, but some customers experience 5 failures in an hour.
- Tell ’em to stop being such whiners.
- Poor performance? More memory and a faster processor!
- What’s wrong with quad Itaniums, 8 Gig of memory, and a liquid cooling system on a $100 SOHO router?

Hard realtime is needed

1. If "rare" timing failures are serious.
2. If precise timing makes a process possible.
3. If precise timing makes a performance advantage.

Timing

1. A 30 microsecond delay can drop 20 Gig Ethernet packets.
2. 10 70hz video streams need packets unloaded at millisecond rates.
3. Time for 1 degree error on manipulator: 10 microseconds.

Low end examples.

1. Replace a digital joystick with an analog joystick and a sound card.
2. Reduce control loop times by sampling A/D at 100 microseconds.
3. Log data over the network using Linux utilities with no software development costs.
4. Use Apache standard web server as a control interface – with no software development costs.

Larger examples.

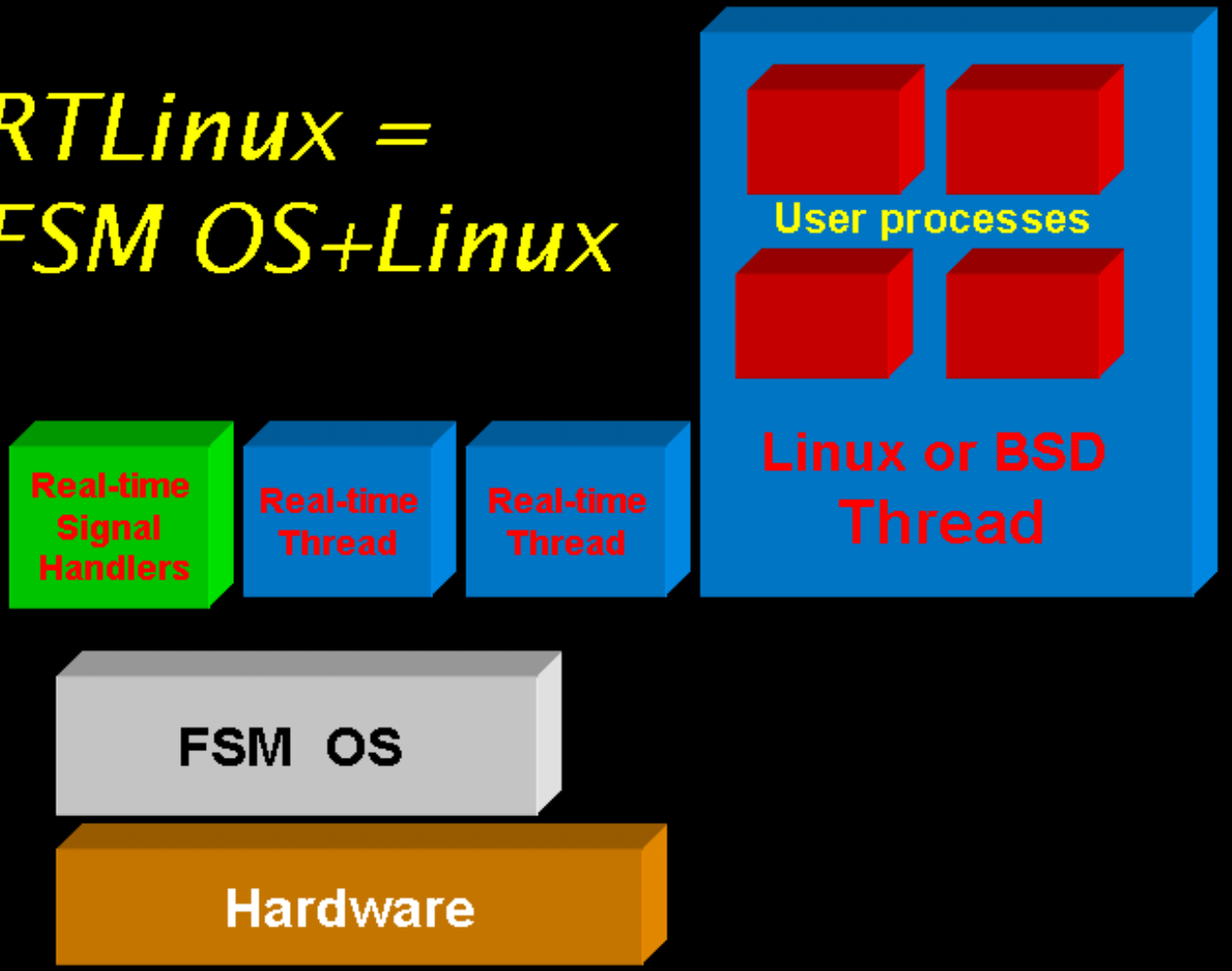
- Multiple software "Virtual routers" operating on packets in realtime.
- Interactive robot control with non-RT graphical interface.

The RTLinux operating system

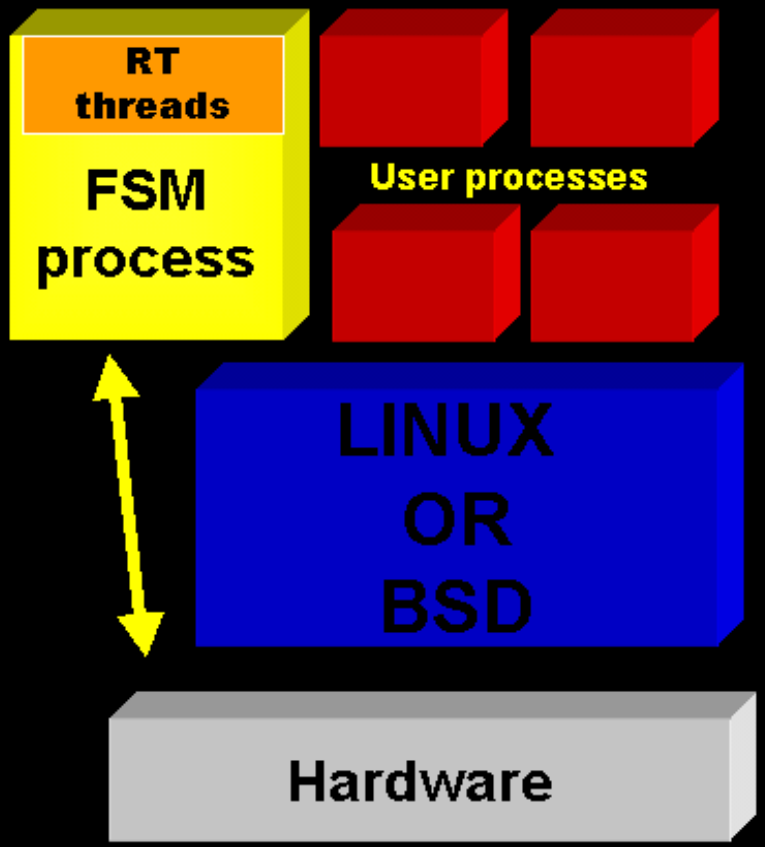
A small hard realtime operating system that runs Linux (or BSD) as its lowest priority task. Used for everything from making chain-saw chains, to switching packets to animating movies.

One view of RTLinux

RTLinux = FSM OS+Linux



Another view of RTLinux



programmer view

Application programmers can treat the FSM OS as a special "real-time" multithreaded process

RTLinux is decoupled from Linux

RTLinux schedules itself — Linux scheduler is not involved.
Linux cannot delay or interrupt execution of RTLinux

Results

1. Example on AMD SC520 133Mhz.
   - *Low latency* response to events. 15microseconds or less.
   - *Precise scheduling* of periodic tasks. 20 microseconds or less error.
2. *On a standard PC: 19 microseconds and 50 microseconds*
3. *On SOCs we are in the single digits*

Programming model

1. For Realtime tasks and event handlers: POSIX Pthreads.
2. For Linux processes connection via POSIX I/O, shared memory, and signals.

As standard as possible with no efficiency loss.

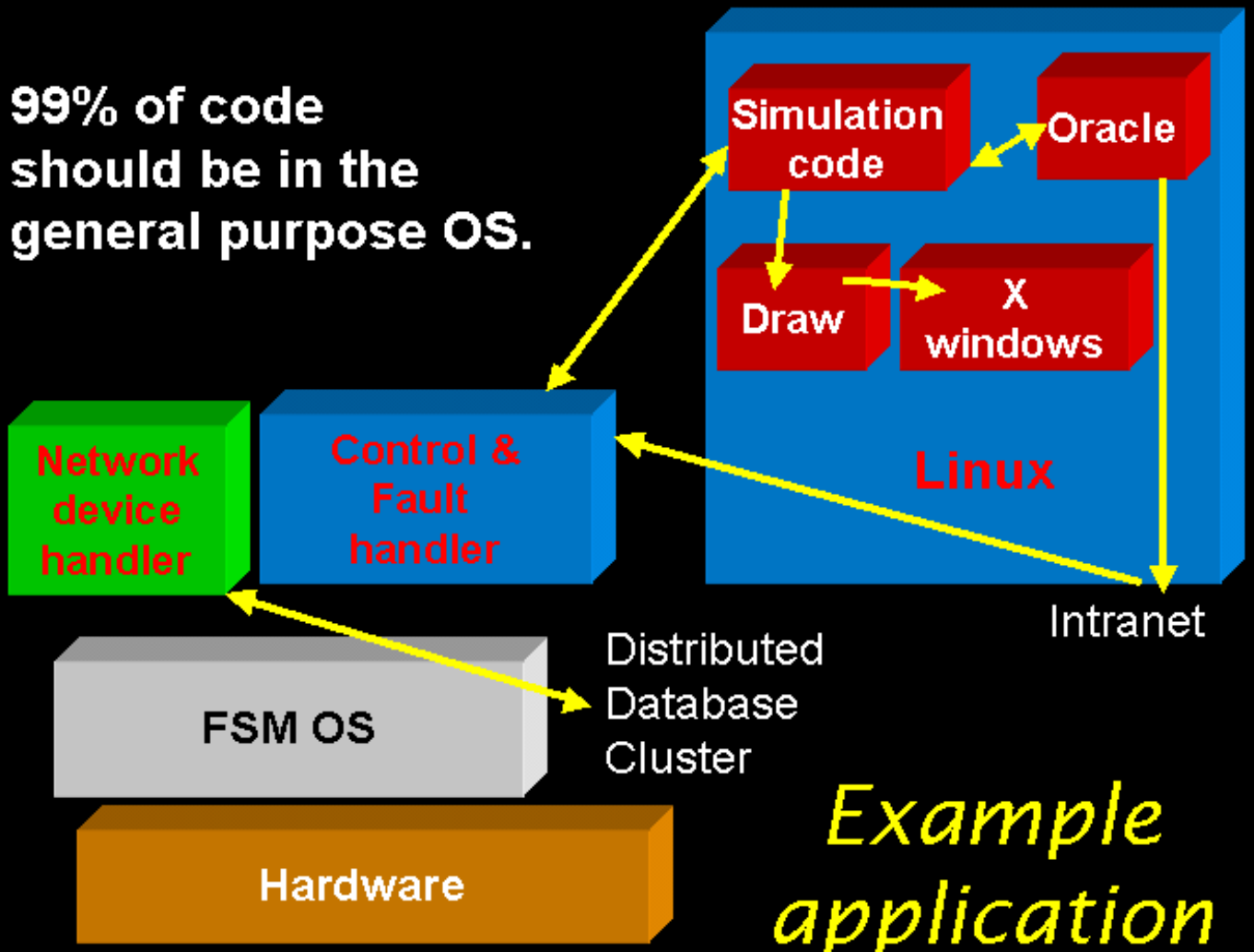
Programming view

1. The hard realtime component of realtime applications should run in a simple, minimal, predictable environment.
2. The non-realtime components should run in UNIX until something better is invented.
3. Hard realtime and non-realtime should be decoupled in operation.

Programming interface

1. "Lean" POSIX threads for RT components, standard POSIX for non-RT components.
2. RTLinux can support alternate APIs.
3. API is programmer convenience: not technology fundamental.

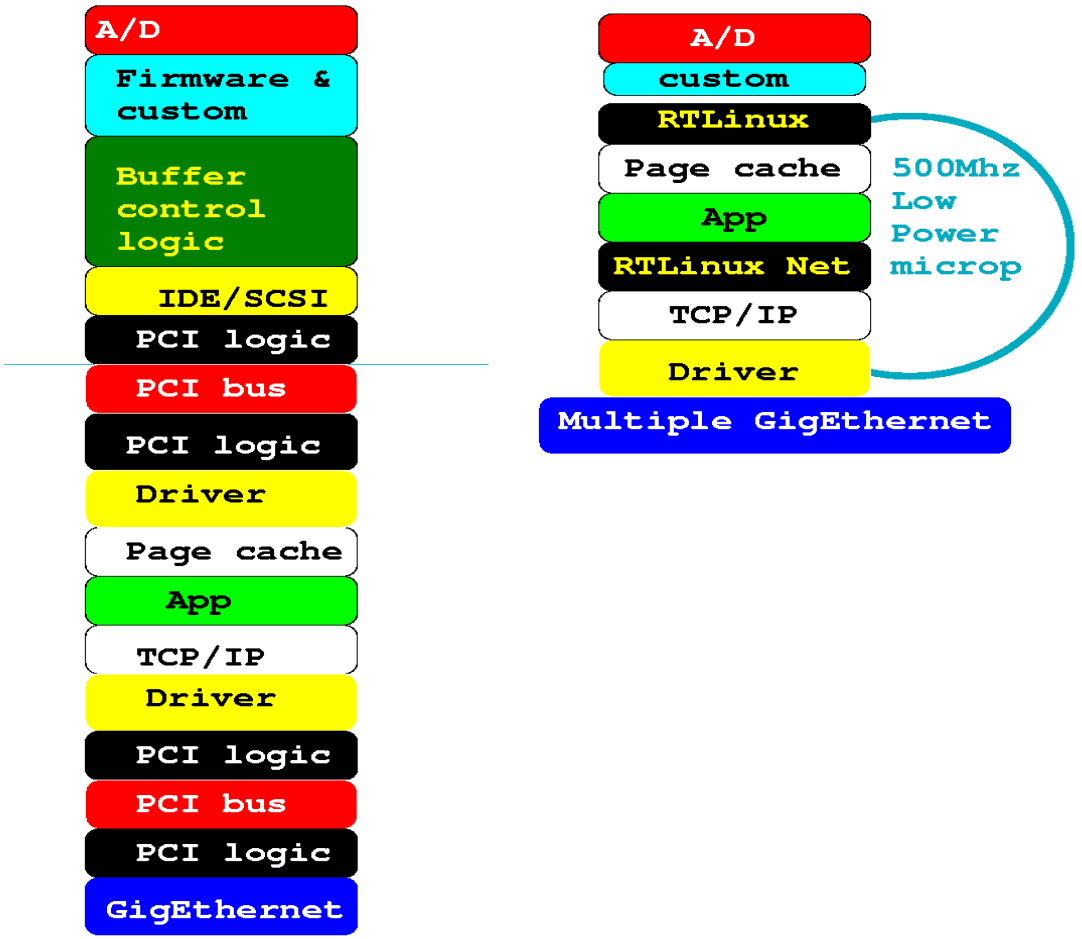
A typical simple application



The simplest user side data logging program

```
cat < /dev/rtf0 > logfile
```

How to use RTLinux



Legacy vs. RTLinux disk drives

Where is it used?

1. Communications: PBX's, routers, ...
2. Factory automation: machine tools, production lines.
3. Robotics: stepper motors, A/D, ...
4. Multimedia: animations, ...

RTLinux Version 3.0

1. V1 was a research project.
2. V2 was the first production version.
3. V3 is the first industrial strength version. x86, Alpha, PPC, MIPS, smp support, ...
4. V4 on the way.

But why not integrate RT into Linux?

- Experience shows: performance limits and engineering costs.

Why is decoupling so important?

1. Mars Lander almost broke because a low priority non-realtime process was able to lock a resource needed by a critical realtime task. The lock was hidden in complex code shared by all tasks, realtime and non-realtime.
2. RTLinux makes all interactions between RT and non-RT explicit and *transparent* .

The RTLinux technical synergy.

1. Highly efficient realtime.
2. Connected to a reliable and powerful networked operating system (Linux).
3. Running on standard PCs, servers, and embedded hardware.

The RTLinux cost advantage.

1. The costs of maintaining general purpose software – data bases, networks, graphics, development, ... – are shared with server and desktop companies.
2. Prototype on PCs.
3. Access to source is essential for important applications.

Coming in V4

1. RT networking. (for industrial control and comms).
2. User mode RTLinux functions.
3. Failover technologies.
4. Optimizations.

Finite State Machine Labs

1. Core kernel development and GPL and non-GPL RTLinux distributions.
2. Training, engineering services and product support.
3. Application software in communications and factory automation.

Why non-GPL development

1. Some of our customers need it: niche products or close hardware/software interaction is common in embedded.
2. We need it: we cannot pay for development on contract services.
3. Our non-GPL is source to customers, but no rights to remarket our code.

www.rtlinux.com