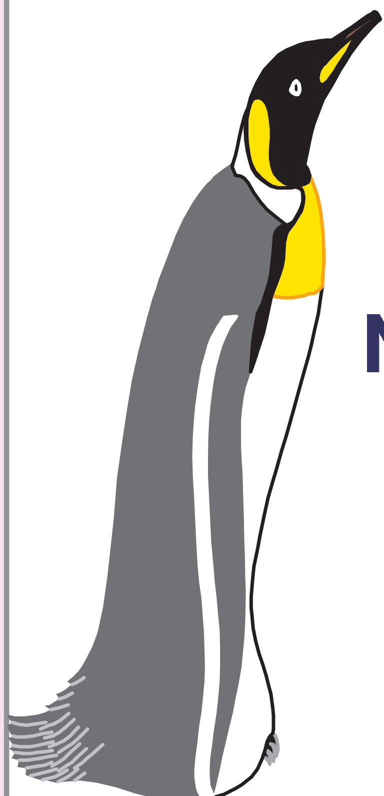
RIDING HURRICANES and STOPPING HEARTS WITH REALTIME LINUX (RTLinux)



Victor Yodaiken
New Mexico Institute of Mining and Technology
Socorro
New Mexico
USA

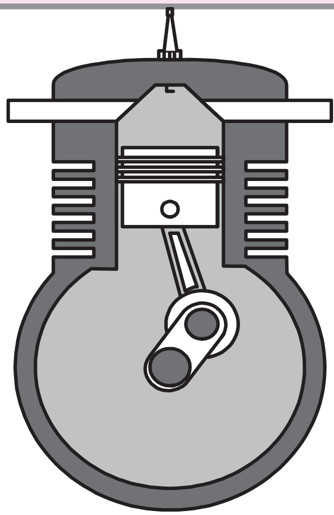
NASA's N43RF

What is realtime?

- Software that interacts with the "real" world outside the computer
- Machinery
- Instruments
- Anything that needs responses in bounded time.

Realtime is often used as a marketing word but it does mean something

- Soft realtime: Tasks need to meet deadline "most of the time". Example: video display.
- Hard realtime: Tasks that do not meet deadlines fail. Example: rocket control

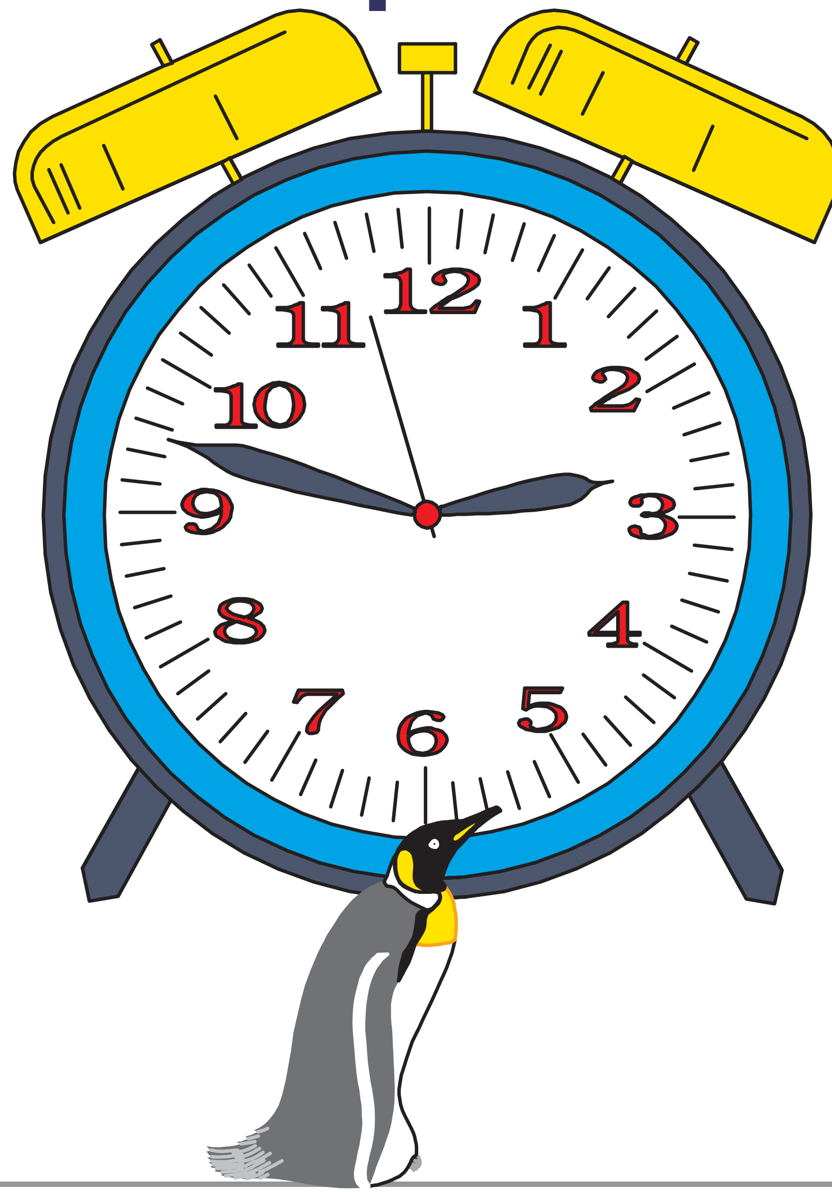


Hard Realtime

- Predictability is key. We need to know what is the imprecision in scheduling.
- Worst case performance is more important than average case.
- In the CS academic literature we say "realtime does not mean fast", but that's only partly true. Speed determines possible range of applications.

Realtime is ubiquitous

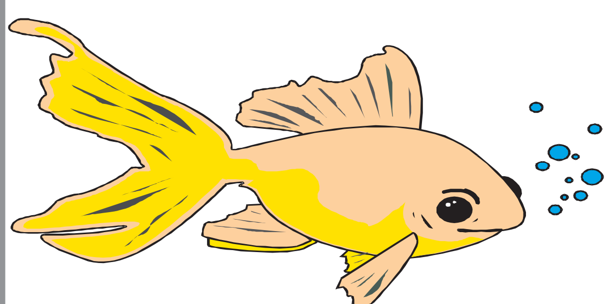
- ❑ Control of scientific instruments
- ❑ Robotics
- ❑ Communications (e.g. SS7 and Frame Relay)
- ❑ Multimedia
- ❑ Machinery (e.g. automobiles)



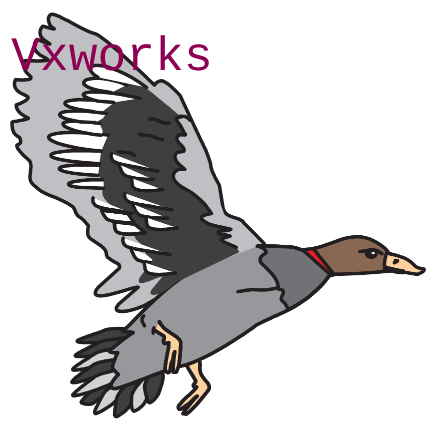
Evolution of RTOSs

- The first realtime OSs were custom, small, simple, and didn't do much.
- But now users want realtime and TCP/IP, Windowing, development, scripting ....

Custom systems



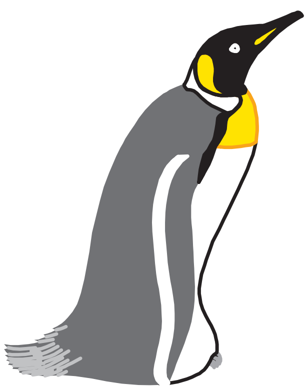
Vxworks

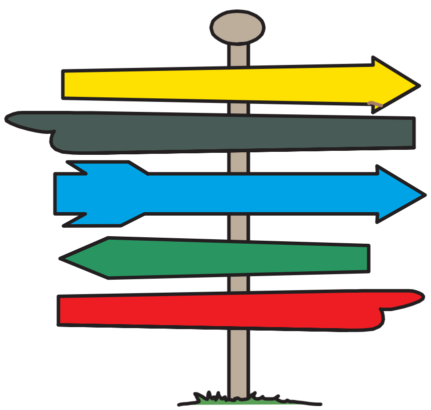


RTLinux

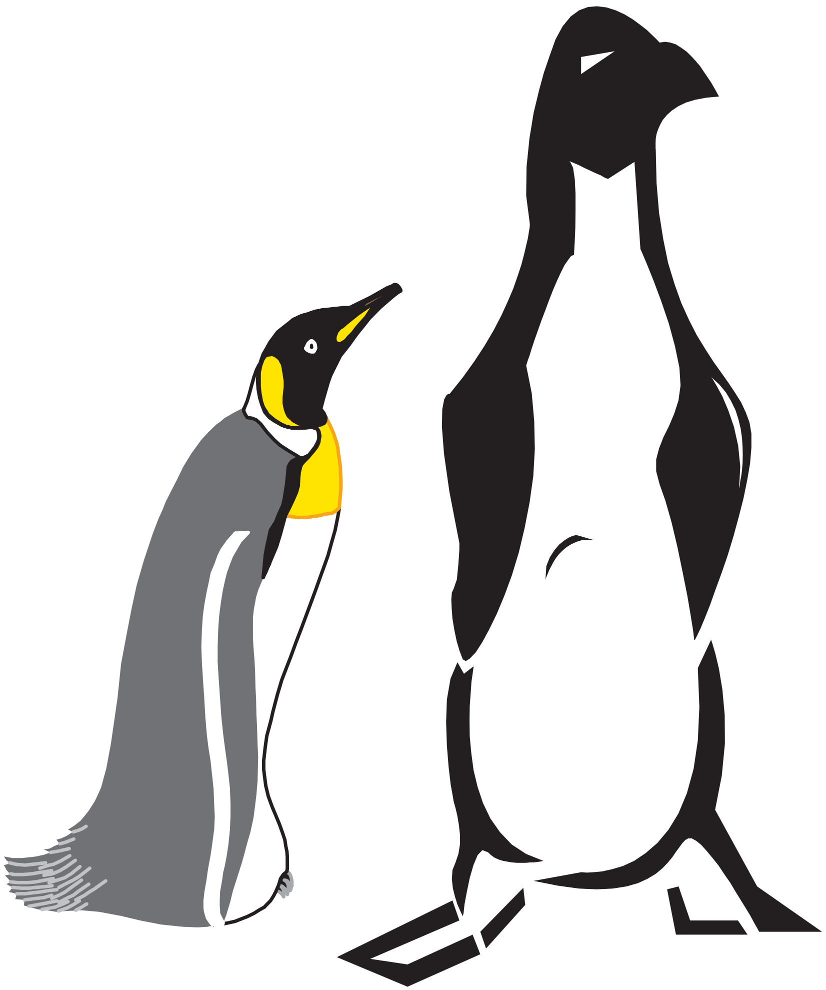
So we want realtime and non-realtime at the same time

- The problem is, as usual, that we want two contradictory things
- Fast average case performance on a OS with everything
- Fast worst case performance on a simple OS.





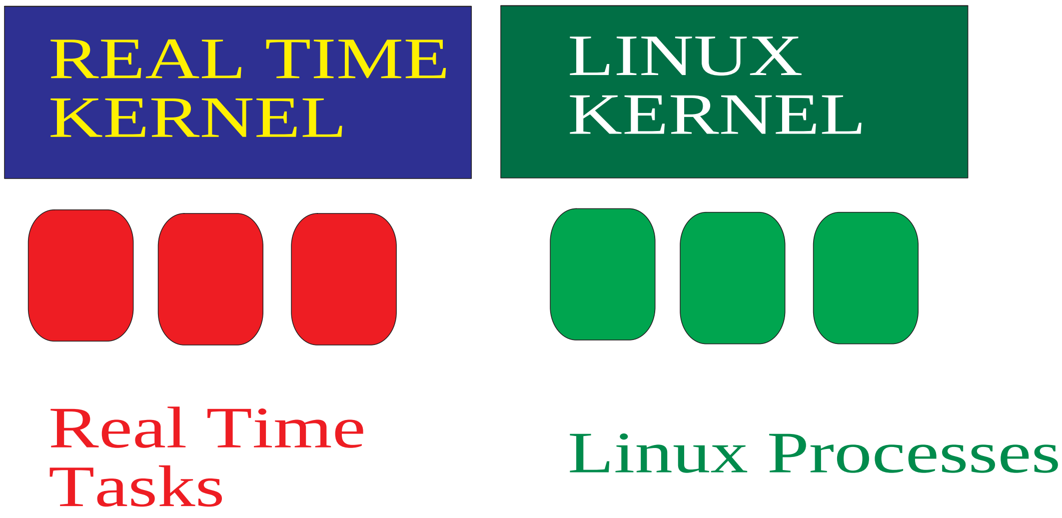
Real Time Linux



- Shares the CPU between the Linux kernel and the Real-Time Kernel
- Allows programmers to split RT and POSIX-style components of applications.

What is it?

- A patch to Linux that adds a co-kernel.
- The co-kernel runs real-time tasks.
- Realtime tasks share the CPU (or CPUs) with Linux and Linux processes.

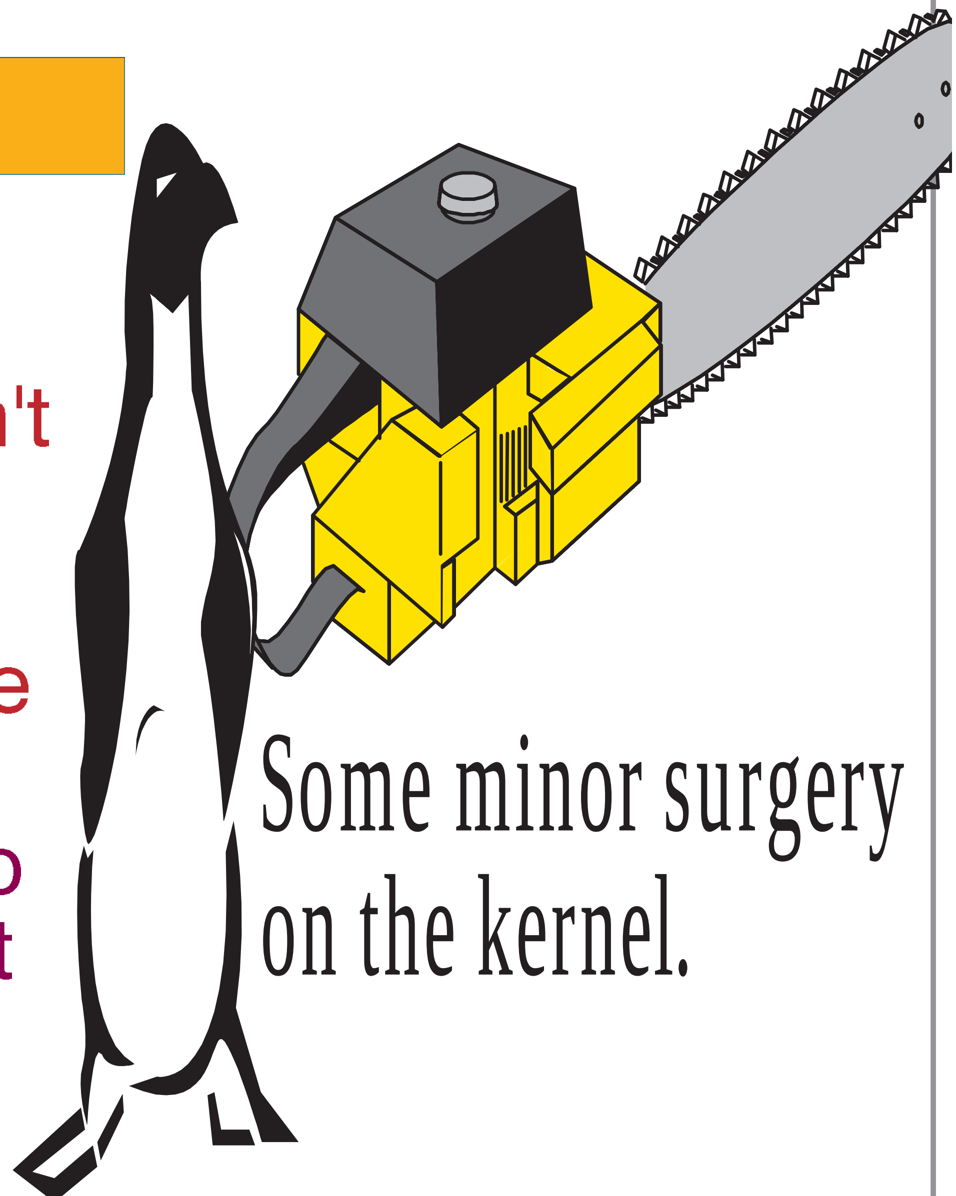


What contradiction?

- RTLinux offers realtime tasks in a primitive, predictable, fast, simple environment and ...
- Connections to Linux processes that have the standard Linux environment available to them.

How does it work?

- "Fix" Linux so it can't disable interrupts.
- A small RT kernel shares kernel space and gets irqs first.
- The main change to Linux is an interrupt control emulator.



"Virtual machines"

- Technique used in the 1960s in MVS
- The MERT OS from Bell Labs [Bell Labs Technical Journal 1978]

Software structure

- Some of the RT Kernel is a patch.
- Most is in loadable kernel modules
- One of those modules provides a device called a rt_fifo that can be accessed by Linux user processes.

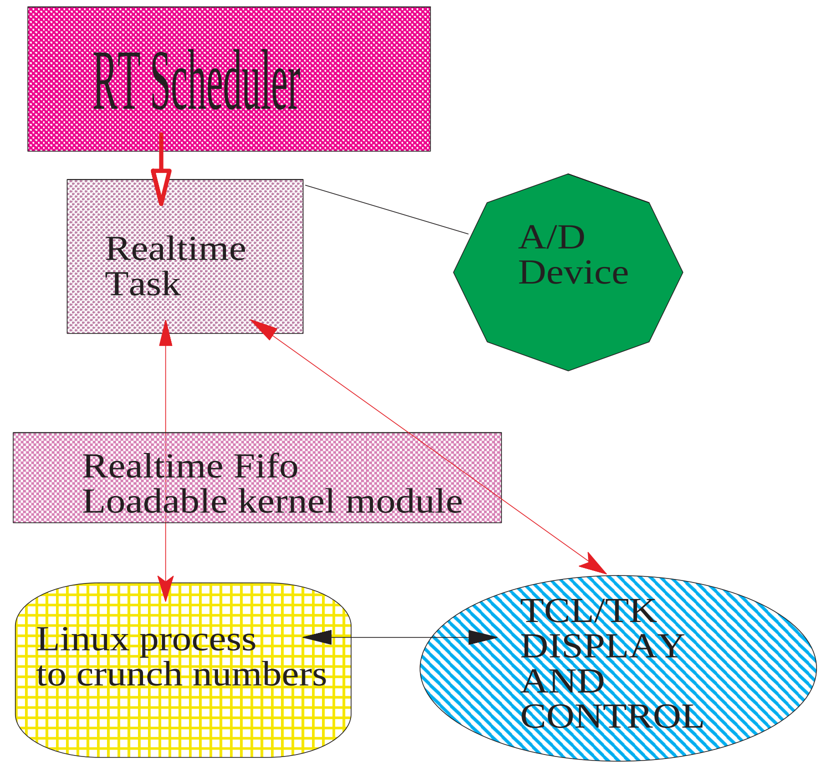
```
fd = open("/dev/rtf1",2);
read(fd,&buffer,100);
/* data from realtime task is in buffer*/
```

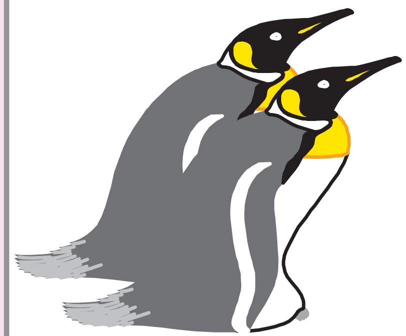
Why can't you have an integrated RT/Non-RT kernel?

- You can --- but you have to pay the price
- To make Linux truly preemptive, you have to throw out 90% of the drivers and rewrite much of the core code. You do not have Linux when you are done.
- Otherwise, worst case delay is longest path between two preemption points and that is not easy to determine or to reduce and every code change has a global effect.

Typical application

- Couple of RT tasks in a loadable kernel module
- Scheduler LKM
- C & Fortran number crunch
- TCL/TK or equivalent



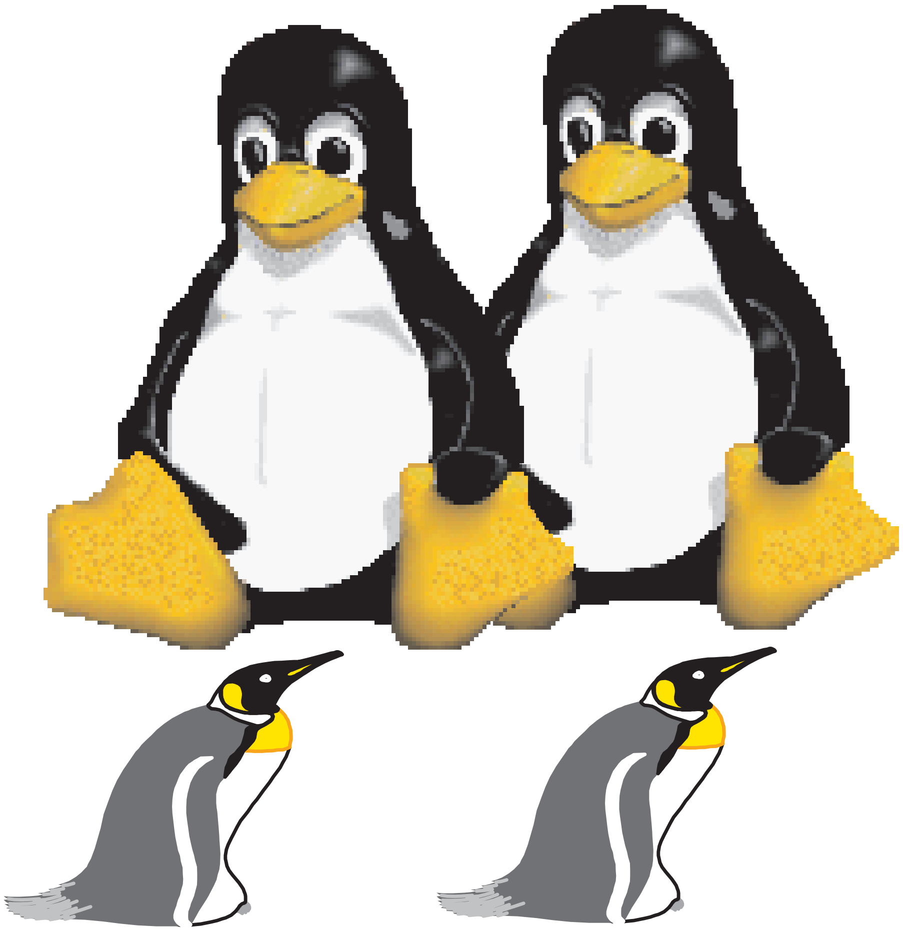


Performance

- RT-Linux is now on Intel machines only, but has been rewritten to facilitate porting. Runs on SMP x86s.
- Under 30us interrupt latency on a 486/33mhz
- 17us worst case delay for a RT periodic task on a P166 running netscape.
- 4us worst case interupt latency on a 233Mhz PII (reported).

SMP

- SMP RTLinux runs on 2.1x Linux kernels.
- Stable on 2 processor machines.
- Tasks can be assigned to processors.



What is it used for?

- Data acquistion: everything from NASA mapping to recording data in physiology experiments.
- Robotics: Mobile robots at Electromechanical labs in Tokyo.
- Machine tool control: NIST has its own RTLinux release.
- Minicams, realtime communications, ...

De-mining robots

- Use Real-Time Linux to collect data and control actuators
- Use Linux for connectivity and back-end processing.

Real-Time is not compatible with standard software design practice

- There are hard limits.
- Average case behavior is not determinative.
- Dynamic resource allocation is dangerous.
- Batching operations is dangerous.
- Hiding complexity is very dangerous!

MARS LANDER and the wrong lessons.

- The Mars "pathfinder" mobile robot stopped working and needed to be patched from earth!
- The problem was "priority inversion" :
  - Low priority process A starts reading IPC
  - High priority B trys IPC, blocks on semaphore.
  - Medium priority C runs so A can't
  - B times out!

How did they fix it?

- Switched on "priority inheritance" algorithm so that A "inherited" B's priority.
- There is a huge CS literature on priority inheritance [Liu and Layland, Sha et al]
- Problem:
- The problem is not solved by this algorithm -- just made less likely.
- The real problem is using non-rt mechanism in an rt system.

Principles of RTLinux design

- The RT kernel should be small, fast, extensible via LKMs.
- If you need sophisticated services, use the Linux side (no dynamic resources on the RT side).
- Use the UNIX model of linking existing code to make new applications.
- Application driven kernel design

What's next?

- More SMP and better response time.
- Realtime clusters and clusters using realtime.
- Embedded systems.
- Incorporation in the main kernel distribution.
- POSIX RT (sort of)
- PowerPCs and maybe Alphas, ARMs ..
- RealTime Perl