

RTLinux/GPL - Introduction

Nicholas Mc Guire
Distributed & Embedded Systems Lab
Lanzhou University, China
<http://dslab.lzu.edu.cn>

Schedule

- Some definitions
- RTOS Design Dilema
- The Dual Kernel approach
- Roadmap
- Conclusion

Some Definitions

- Hard Real-time
- Soft Real-time
- Non Real-time
- Virtualization

Design Dilema

- RTOS should be small and compact
- RTOS should be simple and easy to validate
- Customers want TCP/IP on web servers
- Customers want the latest and hottest video card in the system

So how can one approach these demands ?

Expand the RTOS to a GPOS

- add you can't reuse existing software easily
- restricted capabilities of the RTOS
- no optimizations
- License problems if you want to utilize OSS

Expand the GPOS into an RTOS

- System complexity beomes too high
- testability problems
- you must modify existing packages to give guarantees
- Unrestricted capabilities

The dual kernel concept

- run two kernels concurrently
- virtualize interrupts
- run the GPOS as a task of the RTOS

This concept is fairly old - MERT in 1974 used this in telekom equipment !

What happens to Linux ?

- Linux is one thread of the RTOS only
- Linux has the lowest priority in the system
- Linux is the idle task
- Linux is unmodified internally !

De-facto this is a paravirtualized Linux.

The role of RTLinux

- provide high-resolution timers
- provide a separated execution context
- handle all RT events
- pass on everything else to Linux

RTLinux/GPL is a minimum POSIX environment following POSIX PSE 51 Standard

Roadmap

- 3.2 is the last 2.4 based dual kernel
- 4.0 will switch to XtratuM nanokernel
- RTLinux is targeting POSIX PSE51 and expanding to PSE52
- Distributed RT capabilities are being added
- The API will not change in the future
- Expansions will be POSIX compliant

Conclusion

- RTLinux is a stable RTOS for Linux
- It is free software and it will stay free software
- keep your mind open about alternatives - RTLinux is not the answer for all RT systems

IF you want to try it www.rtlinux-gpl.org .

Advanced Tools Overview for GNU/Linux

Nicholas Mc Guire
Distributed & Embedded Systems Lab
Lanzhou University, China
<http://dslab.lzu.edu.cn>

Tools for GNU/Linux

- problem statement
- tools overview
- development life-cycle
- conclusion

tools are one of the strengths of GNU/Linux - evaluate the capabilities to see if it fits your project needs.

Problem Statement

- resource limitations
- no user monitoring
- applicaton scope insufficient
- failure analysis mandatory

Embedded systems are closer to clusters and servers than to desk-top systems - adjust your debugging !

Problem Statement

- resource limitations
- no user monitoring
- applicaton scope insufficient
- failure analysis mandatory

Embedded systems are closer to clusters and servers than to desk-top systems - adjust your debugging !

Kernel Space Tools

- GDB/KGDB
- KFI/KFT
- GCOV/Kernel GCOV
- Oprofile
- LTT/LTTng
- Kprobes
- Kernel builtin debug extensions
- /proc interface

User Space Tools

- strace/ltrace
- LD_PRELOAD
- valgrind
- BGCC
- standard unix tools

GDB/KGDB

- `/proc/kcore` to check the running kernel
- KGDB to debug the kernel

GDB will only help you if you have the experience of using it in the kernel - plan it in in your software development life cycle

KFI/KFT

- -finstrument-functions
- set config via `/proc/kft`
- get data via `/proc/kft_data`
- 50-100% overhead !
- powerfull tool to understand the kernel

KFI/KFT will be helpfull in finding complex bugs - but only if you know how to use it before you need it !

GCOV / Kernel GCOV

- spanning tree of the kernel functions
- 64bit event counters
- interface via `/proc/gcov`
- system and kernel level performance assessment tool

GCOV is a standard profiling tool extended into kernel space with kernel GCOV - it is also available for UML

Oprofile

- built on performance counters (PMC)
- X86 and PPC
- precise for low-level HW-units
- low overhead

Oprofile is primarily used for performance tuning and locating of hardware artefacts (i.e. cach thrashing)

LTT/LTTng

- static instrumentation
- data via relayfs
- gives good system level overview
- good for detecting application interaction problems
- low overhead
- X86 centric but PPC port available

Kprobes

- breakpoint debugging in kernel mode
- insertion of arbitrary handlers
- low overhead if used carefully
- relatively complex to use

Kernel Builtins

- scheduling statistics
- preempt timing measurement
- lock dependency checker
- vm debugging options

/proc Interface

- 0-overhead
- easy to use
- easy to integrate into user interface
- highly specialized information only

SW-Lifecycle Issues

- plan in tools in your regular test procedures
- if you don't know the healthy system you can't read the pathological case !
- log your test and debug sessions !
- plan in the time for learning tools - you can't start using them when you need them

SW-Lifecycle Issues

- go to the target as late as possible
- keep your code arch independant by testing on two platforms
- automate the usage of Linux kernel debug tools
- integrate the tools into your product so you can get hig quality bug reports from your customers

Not every product will need this - but many more than currently are using the capabilities of GNU/Linux

Conclusion

- Learning GNU/Linux tools is an investment
- GNU/Linux tools allow better inspection than most commercial tools
- The GNU/Linux tools cover the entire spectrum from the application layer into the kernel down to the hardware
- Dont wait to learn them until you need them !