Real-Time Systems

Challenges using Linux to implement Real-Time Services

Dr. Sam Siewert

Electrical, Computer and Energy Engineering

Embedded Systems Engineering Program

Overview

- Review of Linux approach compared to RTOS and Cyclic Executive
- Linux Kernel Space – Option #2 (Why not just use an RTOS!)
- Linux User Space – Option #1 (Has advantages, but high overhead)
- Why User Space is good
- Why User Space is bad
- How to use a combination of User and Kernel space with RT configuration

3 Approaches to RT Systems with Software Sevices

Cyclic Exec	RTOS	OS + RT POSIX
E.g. Shuttle Flight Software, Network elements, process control, Ada83/95 CE	VxWorks, QNX, ThreadX, TI RTOS, FreeRTOS, Zephyr, Nucleus, RTEMS, etc.	RT Linux, Solaris, LynxOS, FSM Labs, Concurrent, RTAI, Linux Foundation RT, etc.
Custom, Deeply Embedded Systems	Embedded Systems, Scalable and Portable	RT Services for Scalable Apps and Systems
Low over-head, purpose-built	Medium overhead, quick to market	Highest overhead, mixed RT + SRT + BE
Costly to develop	License costs	Maintenance costs

Linux User Space vs. Kernel Space

- Linux has 2 address spaces – user and kernel
  - Kernel space directly interfaces to hardware and manages all resources
    - Provides protected address spaces for user processes (that can have threads mapped onto kernel tasks)
    - Provides security and stability
    - Driver and kernel modules have different development, debug, and test strategies!
  - User space protects the system from errors and provides security & stability
    - All applications interfacing to hardware must go through a driver and/or system call
    - All applications run in within a protected address and resource domain
  - Can we have both? (cake and eat it too?) – with care
    - Sub-millisecond time critical, direct hardware interfaces must be in a driver kernel module
    - Millisecond or less predictable response (soft RT and BE) services can operate in user space
    - Carefully configure a Linux platform (real-time distro) and allocate services to both user space and kernel space based on requirements and timing constraints (carefully!) – TEST it!

AMP Fixed Priority RM Policy on Linux

- Linux is SMP by default on Multi-core systems
  - We emulate AMP with Pthread Affinity
  - We implement Fixed Priority RM Policy with SCHED_FIFO
  - Check IRQ balance (irqbalance) and consider masking off RT cores

- We need periodic source of interrupts, driver, and sequencing (sub-millisecond services must be implemented in kernel space in a driver module)

- Without a periodic source of interrupt, we must sequence operations with timers

- User space RT services are possible, but accuracy and precision impacted by User to Kernel system call overhead (accurate to 1 to 10 milliseconds at best!)

Linux Kernel Patches and Configuration for Real-Time

- Linux is not configured for Real-Time in most distributions!
  - Notable exceptions: Automotive Grade Linux, Concurrent RedHawk Linux, Wind River Linux, etc.
  - We can purchase support for pre-configured RT Linux distro or roll our own

- Key patches recommended (not used in this course for simplicity)
  - RT Preempt patch for Kernel
  - RT Mutex priority inheritance semaphore patch (or use of FUTEX semaphores)
  - High Resolution Timers (enable in user space – TSC on x86/x64, STC on ARM) or keep time in kernel space

- Key configuration updates (not used for simplicity)
  - run levels – make your own for your services!
  - remove unused services, pair down features to just those required!

Avoid Large “Unknown” Libraries in Linux

- Attraction to an OS like Linux is code and platform reuse!
  - Up and running quickly, working on drivers and applications
  - Safe for multiple developers
  - Stable with security features
- Challenge is that short of reviewing all source code use, we don’t know what is really going on!
- Pair down the system to essentials and use custom RT run-level
  - E.g. run-level “4” is for “special purposes”
- If large libraries are used – e.g. ROS, OpenCV, etc., then review source and test!!

What we do in ECEE 5315 – 5318 Coursers Series

- We use the Raspberry Pi for convenience and simplicity
- We learn HRT, SRT, and Best Effort service theory! – useful for CE, RTOS or Linux
  - We practice implementation of RT Services as POSIX SCHED_FIFO threads
  - We work around accuracy and precision limitations in user space (millisecond accuracy is sufficient for our learning objectives – 30 Hz or less)
  - We learn about Linux kernel space options, RTOS alternatives, ARM M Series or R Series Cyclic Executive options
- At the end, as a new RT theorist and practitioner, you will decide – CE, RTOS or Linux + RT configuration and extensions?
- Optional Textbook – RTECS with Linux and RTOS covers both options
- When in doubt – go with RTOS, CE or Hardware HRT Services!

How many systems are HRT?
(loss of life, property, or $$$)

- Civil aviation flight control – YES!
- Medical ICU equipment – YES!
- Public transportation – YES!
- Energy generation, distribution and monitoring – YES!
- Self-driving car, Autonomous Air taxi – YES!

But, many things are not …
(loss of quality, not system, low or no collateral damage – safety judgement)

Robotics? - Maybe (size, power)
Digital Media – SRT QoS
Games – SRT QoS
UAV/UAS – Some HRT (autopilot), some SRT (navigation and imaging)
Driver's assistant – navigation, warnings and specific functions