Why Linux is not an RTOS:
porting hints

Chris Simmonds
2net Limited

*Embedded Systems Conference UK. 2009*

Copyright © 2009, 2net Limited
Overview

- Linux is a popular choice as an embedded OS
- Most projects evolve from previous projects often based on an RTOS
- How to get from point A (RTOS) to point B (Linux)?
  - Is Linux an RTOS??
Porting options

(*) for example v2lin [1] or Mapusoft
The porting dilemma

• How much existing code can I keep?
• How much effort is required to port my application to Linux?
• What should I look out for?
• What are the gains?
Why Linux is not an RTOS

- Applications run in “user space”
- All hardware interaction is in “kernel space”
- All i/o via files and sockets
- Applications are processes
- Default scheduling policy is time shared
- POSIX API
- Is Linux real-time?
Different memory models

RTOS: all one memory space

Linux memory spaces

User space

Kernel space

Application

C library

Linux

RTOS

Hardware

Hardware

Application

C library

Linux

RTOS

Hardware

User space

Kernel space
Device drivers

Linux

do_IRQ()

Device driver

ISR

Hardware
Why Linux is not an RTOS

• Applications run in “user space”
• All hardware interaction is in “kernel space”
• All i/o via files and sockets
• Applications are processes
• Default scheduling policy is time shared
• POSIX API
• Is Linux real-time?
Everything is a file

For example, to read characters from first serial port, open device node (file) /dev/ttyS0

```c
f = open("/dev/ttyS0", O_RDWR);
l = read(f, buf, len);
```

/dev/ttyS0

Linux

UART driver

4:64

Major number 4
Minor number 64
My device doesn't look like a file!

- File read & write operations work well with byte streams - such as a serial port
- How about a robot arm?
- The ioctl function allows any interaction you want

```
struct robot_control_block rc;
f = open ("/dev/robot", O_RDWR);
...
ioctl (f, SET_ROBOT_PARAMETERS, &rc);
...```
Hint 1

• Identify all code that accesses hardware directly
• Design a file-based interface
  • Use ioctl for things that do not naturally fit the file concept
• Make this into a device driver
• Remember: keep device drivers as simple as possible
  • All the complicated stuff should be in the application
Cheating: user space drivers

- mmap allows an application direct access to device memory
  - But, cannot handle interrupts
  - No control of CPU cache or instruction queue
  - Not the “Linux way”
- The Linux User IO subsystem uses mmap to provide a flexible framework for user space drivers
  - UIO [2]
mmap example

```c
#include <sys/mman.h>

#define IO_PHYS_ADDRESS 0x90600000
#define ARM_POS 0x0034
#define ARM_MOTION 0x0038

int main (int argc, char *argv[])
{
    unsigned led_dat;
    int mh;
    char *ptr;

    mh = open ("/dev/mem", O_RDWR);
    ptr = mmap (0, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED,
                mh, IO_PHYS_ADDRESS);

    *(unsigned int*)(ptr + ARM_POS) = new_pos;
    while (*(unsigned int*)(ptr + ARM_MOTION) != 0)
        sleep (1);
    ...
```
Why Linux is not an RTOS

• Applications run in “user space”
• All hardware interaction is in “kernel space”
• All i/o via files and sockets
• Applications are processes
• Default scheduling policy is time shared
• POSIX API
• Is Linux real-time?
Processes

**Process =** address space + program + thread of execution

Some process have > 1 thread
Pros and cons of processes

• Pro
  • Protected memory space
  • Resources (memory, open files) released when exit
  • Easy to re-start a failed process

• Con
  • Communication between processes quite slow & cumbersome
Pros and cons of threads

• Pro
  • Easy communication using shared variables, mutexes and condition variables
  • Similar memory model to RTOS tasks

• Con
  • No memory protection between threads in the same process
Why Linux is not an RTOS

- Applications run in “user space”
- All hardware interaction is in “kernel space”
- All i/o via files and sockets
- Applications are processes
- Default scheduling policy is time shared
- POSIX API
- Is Linux real-time?
Scheduling policies

• **SCHED_OTHER**
  • Time share: fairness. Priority set by scheduler

• **SCHED_FIFO**
  • Fixed priority (1 to 99); preempts SCHED_OTHER
  • Use this for real-time activities

• **SCHED_RR**
  • As SCHED_FIFO but tasks of same priority time-slice
  • Default quantum is 100 ms
Hint 2

- Make closely-coupled groups of RTOS tasks into POSIX threads within a process
- Separate RTOS tasks with little interaction into separate processes
- Make real time threads SCHED_FIFO
  - Use Rate Monotonic Analysis or similar to choose priorities [3]
- All non real-time threads should be SCHED_OTHER
Why Linux is not an RTOS

- Applications run in “user space”
- All hardware interaction is in “kernel space”
- All i/o via files and sockets
- Applications are processes
- Default scheduling policy is time shared
- POSIX API
- Is Linux real-time?
POSIX

• POrtable Operating System Interface
  • IEEE standard 1003.1
• Most RTOS functions map to POSIX one-to-one
  • Tasks to POSIX threads
  • Semaphores and mutexes to POSIX semaphores, mutexes and condition variables
  • Message queues to POSIX message queues
  • Watchdog timers to POSIX clocks and timers
Look out for

• POSIX Threads
  • Threads run immediately they are created
  • Not possible to terminate an arbitrary thread

• POSIX semaphores and mutexes
  • POSIX has many types of mutex, including priority inheritance. See [4]
  • POSIX does have semaphores but they are not much used. See [5] for a discussion on mutexes vs semaphores
Library and kernel versions

• For full POSIX compliance in Linux you need current versions of Linux and the C library
  • Kernel >= 2.6.22
  • GNU C library >= 2.5
• Beware uClibc [6]
  • Small, “embeddable” C library
  • Good for small systems with <= 16MiB storage
  • BUT, lacks many POSIX functions
Hint 3

• You are going to have to re-write some code
• Where possible, re-factor code around shared data
  • Write accessor functions to hide data structure from rest of the program
  • Protect against concurrent access using a mutex
  • In the literature this is called a monitor [7] - makes future maintenance and porting much easier
Why Linux is not an RTOS

- Applications run in “user space”
- All hardware interaction is in “kernel space”
- All i/o via files and sockets
- Applications are processes
- Default scheduling policy is time shared
- POSIX API
- Is Linux real-time?
Is Linux real time?

✓ Deterministic scheduler
✓ Static priorities (SCHED_FIFO)
✓ Priority inheritance mutexes
✓ Lockable memory - stops demand paging
✓ High resolution timers
?
Deterministic interrupt response
Demand paging

- Pages of code and data are read from the program file on demand 4KiB at a time
- Causes jitter in real-time programs

Hint:
You can page in and lock all memory using mlockall (MCL_FUTURE);
Interrupt latency

Interrupt latency

 ISR completes & calls wake_up

Task is scheduled

Interrupt latency

Preemption latency

Interrupts disabled

ISR execute

Time

T₀ T₁ T₂ T₃
Kernel preemption options

• Default - no preemption in kernel mode
  • Good for throughput, bad for real-time
• Preemptible kernel
  • Reduces jitter in preemption latency
  • Good for soft real-time
• PREEMPT_RT [8]
  • Reduces jitter in all three areas
  • Good for “firm” real-time
  • Not in the main line kernel yet
Hint 4

• In a real-time system, work out what deadlines you have and how much jitter you can accept

• Lock memory in any process with real-time threads with mlockall

• For soft real-time with jitter ~ millisecond enable kernel preemption

• For “firm” real-time with jitter ~ 10's or 100's microseconds use the PREEMPT_RT patch
Summary

• Porting to Linux will require some code refactoring
  • Hardware requires device drivers
  • Tasks become threads in one or more processes
  • Map RTOS functions onto POSIX
  • Select real time model
• Is Linux an RTOS?
  • No: it is a complete operating system!
References

[1] v2lin - VxWorks API for Linux
http://v2lin.sourceforge.net/

[2] UIO: user-space drivers
http://lwn.net/Articles/232575

[3] Rate-monotonic scheduling
http://en.wikipedia.org/wiki/Rate-monotonic_scheduling

http://www.embedded-linux.co.uk/tutorial/mutex_mutandis

http://www.feabhas.com/blog/labels/Semaphore.html

http://www.uclibc.org/

http://en.wikipedia.org/wiki/Monitor_(synchronization)

[8] The PREEMPT_RT real time patch series
http://www.kernel.org/pub/linux/kernel/projects/rt/