Threaded IRQs on Linux PREEMPT-RT

Luís Henriques
Intel, Shannon

OSPERT 2009

Agenda

- Threaded IRQs overview
  - Why threaded IRQs
  - PREEMPT-RT overview
  - Threaded IRQs in PREEMPT-RT
- Experimental results
  - Experiments description
  - Experiments results
  - Conclusions

Why threaded IRQs

- Threaded IRQs is a common design-pattern in other operating systems
- Benefits:
  - Increased observability
  - Interaction between interrupt handlers and softirqs/tasklets can be simplified
    - Reduced locking complexity
  - Improve system predictability

- **But:**
  - Overall system throughput can decrease

PREEMPT-RT overview

- Linux is a GPOS kernel
  - Give all tasks a **fair** share of resources
- Latencies depend on everything running on the system
- Main cause: preemption may be switched off for an unknown amount of time
- Thus, Linux **does not** guarantee timing
  - Although it is considered ‘good enough’ for many applications

PREEMPT-RT overview

Main characteristics

- Complete kernel preemption
  - Reduces scheduling latency by replacing most of the spinlocks with blocking mutexes
- High-resolution timers
- Priority inheritance protocol
- Threaded IRQs

Threaded IRQs in PREEMPT-RT
ISRs on Linux

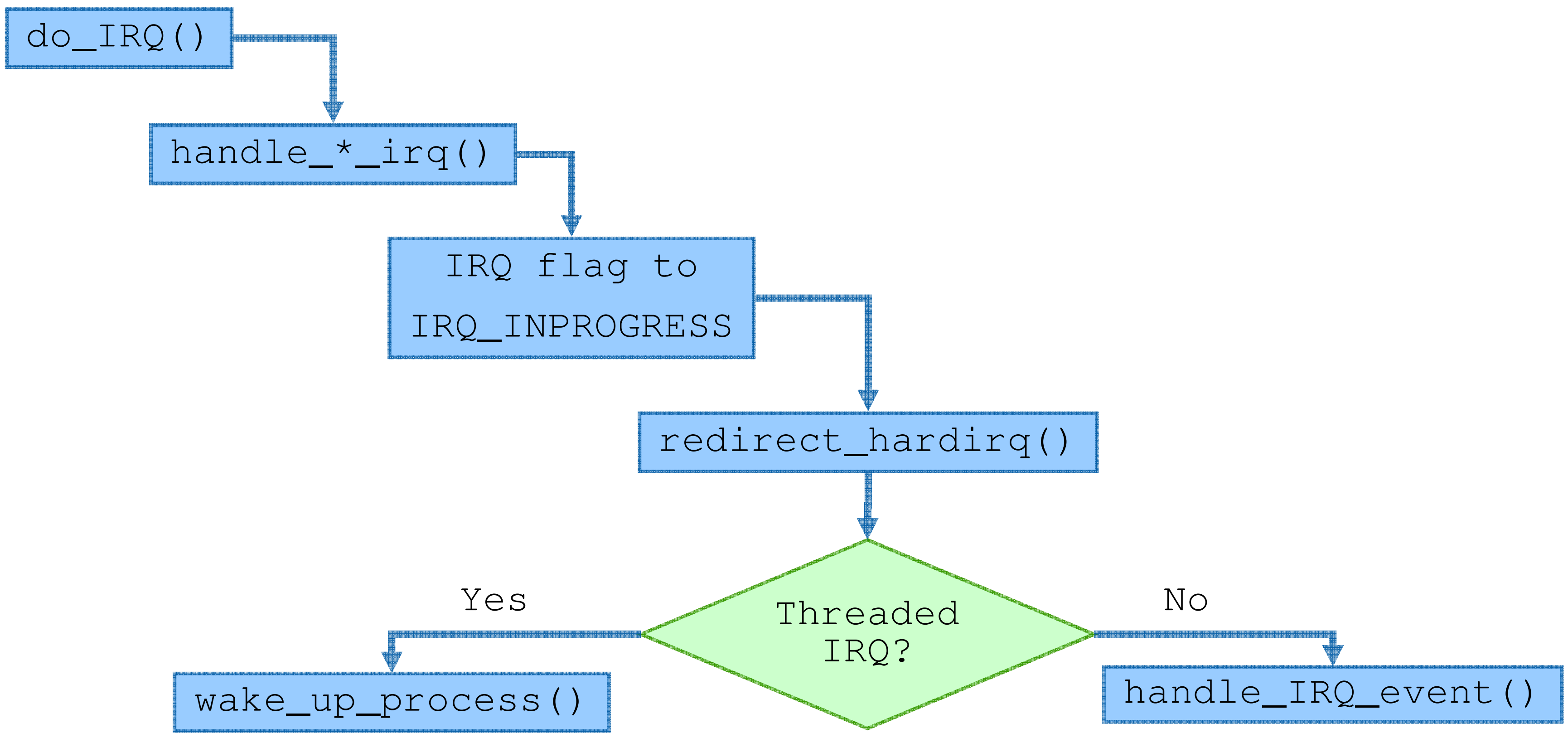
- With mainline Linux, when an interrupt occurs, CPU is preempted and ISR is executed
  - ISR is executed at highest priority
    - Typically with interrupts disabled or current interrupt line masked off
  - ISRs can be preempted only by other interrupts
- A well written device driver:
  - Do very little work on ISR
  - Push time-consuming activities to kernel threads, tasklets or softirqs

Threaded IRQs in PREEMPT-RT
ISRs on PREEMPT-RT Linux

- Device drivers register interrupt handler with usual interface (`request_irq()`)
  - No modifications required in device drivers
- A thread is created for the IRQ
  - Only one thread per IRQ
- Kernel keeps a list of ISRs for each IRQ
  - ISRs are sequentially invoked for shared IRQs
- Some drivers may not want their interrupt handlers threaded (e.g., clock and serial I/O on FreeBSD)
  - `IRQ_NODELAY` flag for non-threaded IRQs

Threaded IRQs in PREEMPT-RT

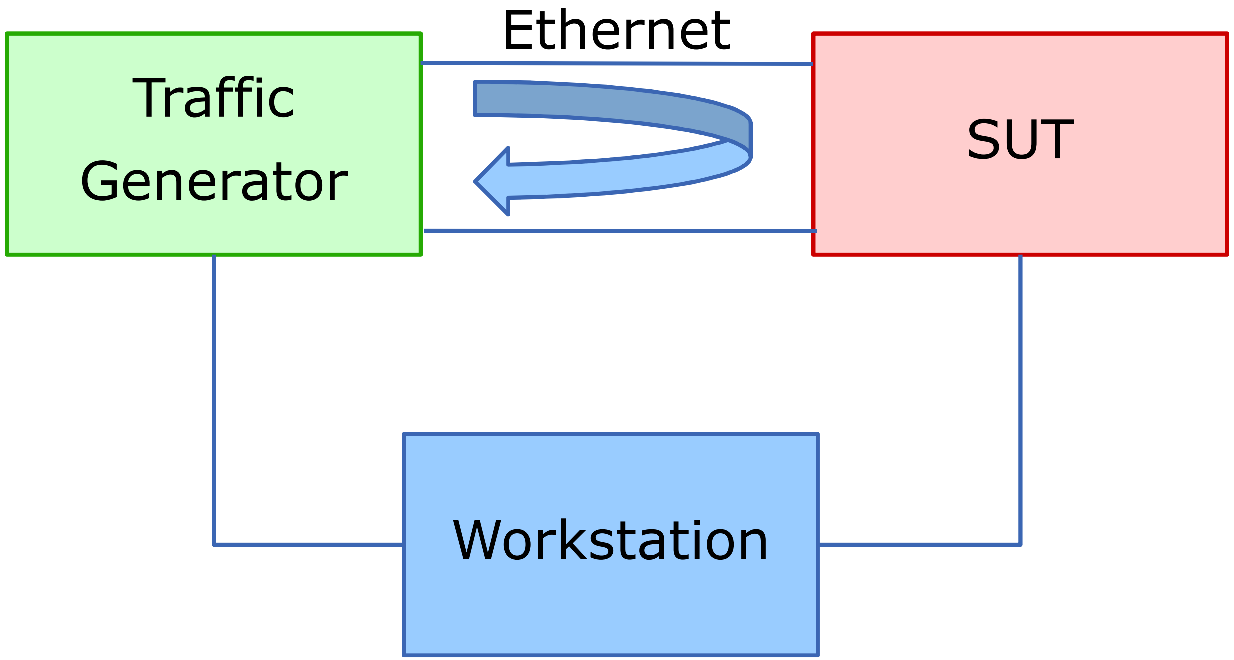
ISRs on PREEMPT-RT Linux



Agenda

- Threaded IRQs overview
  - Why threaded IRQs
  - PREEMPT-RT overview
  - Threaded IRQs in PREEMPT-RT
- Experimental results
  - Experiments description
  - Experiments results
  - Conclusions

Experiments description
Test environment



Experiments description

Test environment

- Hardware:
  - Intel® EP80579 processor
    - UP SoC at 800MHz
  - Intel® 82572EI Gigabit Ethernet
    - PCI Express
- Kernel:
  - 2.6.29.3 and 2.6.29.3-rt13 patchset:
    - Vanilla
    - PREEMPT-RT with Threaded IRQs only
    - PREEMPT-RT with Threaded IRQs + complete preemption
- Cyclictest
  - Measures accuracy of wakeup from sleep (500 usecs)

Experiments description

Test campaigns

- Five test scenarios:
  - Vanilla kernel
  - PREEMPT-RT kernel with Threaded IRQs config options:
    - Cyclictest with low priority
    - Cyclictest with high priority
  - PREEMPT-RT kernel with Threaded IRQs + “Complete preemption” config options:
    - Cyclictest with low priority
    - Cyclictest with high priority
- Traffic injected at several (fixed) rates
  - 64 bytes packets

Experiments results
Vanilla kernel

Traffic Rate (usecs)	cyclictest			IRQs		Lost Frames (%)
	Min	Avg	Max	eth0	eth1	
10	5	48	585	18,076,863	13,882,919	0.00
9	4	48	1,180	13,912,623	11,093,282	0.00
8	4	52	1,051	10,644,941	9,791,348	0.00
7	4	76	1,435	10,350,929	9,970,528	0.00
6	5	94	1,893	7,808,480	8,708,121	0.00
5	3	193	6,553	4,556,491	5,172,545	0.00
0	3	5	89	-	-	-

Experiments results
Threaded IRQs kernel, cyclictest priority LOW

Traffic Rate (usecs)	cyclictest			IRQs		Lost Frames (%)
	Min	Avg	Max	eth0	eth1	
10	6	128,474,238	268,133,338	6,170,139	5,991,550	0.00
9	7	129,139,176	269,522,352	6,055,578	6,000,188	0.00
8	7	136,136,357	274,904,968	5,960,877	5,946,526	0.00
7	9	139,804,614	280,713,676	5,405,771	5,589,278	0.00
6	256	154,922,510	310,819,204	4,239,118	4,350,945	0.55
5	-	-	607,364,076	29,896	26,134	7.39

Experiments results

Threaded IRQs kernel, cyclictest priority HIGH

Traffic Rate (usecs)	cyclictest			IRQs		Lost Frames (%)
	Min	Avg	Max	eth0	eth1	
10	4	8	65	6,223,553	5,994,708	0.00
9	4	8	479	5,780,033	5,836,146	0.13
8	5	9	74	5,123,027	5,277,829	0.00
7	6	135,580,508	272,452,238	4,467,440	4,591,824	0.00
6	6	140,860,392	282,329,933	2,830,294	2,808,786	1.65
5	8	156,278,465	313,641,236	16,384	14,435	9.37
0	4	5	93	-	-	-

Experiments results

PREEMPT-RT kernel, cyclictest priority LOW

Traffic Rate (usecs)	cyclictest			IRQs		Lost Frames (%)
	Min	Avg	Max	eth0	eth1	
10	2	60	1,417	6,094,390	5,887,783	0.00
9	2	74	1,294	6,053,376	5,858,603	0.00
8	7	129	1,466	5,553,851	5,596,537	0.00
7	2	152	1,546	4,810,329	5,044,578	0.00
6	1	1,610	75,579	3,644,729	3,765,491	1.51
5	370,627	252,284,884	496,880,098	1,724,908	1,629,418	5.14

Experiments results

PREEMPT-RT kernel, cyclictest priority HIGH

Traffic Rate (usecs)	cyclictest			IRQs		Lost Frames (%)
	Min	Avg	Max	eth0	eth1	
10	5	9	27	6,158,486	5,835,824	0.00
9	5	9	29	5,792,293	5,617,035	0.00
8	5	9	32	5,558,052	5,573,145	0.00
7	5	9	30	4,774,998	4,898,368	0.00
6	1	42	25,880	3,398,359	3,431,860	0.33
5	5	1,249	50,153	454,401	407,200	6.96
0	4	5	25	-	-	-

Conclusions

- Overview of threaded IRQs on PREEMPT-RT
- Experimental results:
  - Real-Fast: vanilla kernel
  - Real-Time: PREEMPT-RT kernel
    - Packets are lost at higher rates
- Probably, there's still space for optimisations:
  - IRQ threads with same priority sharing same thread
  - No thread if there are no higher priority threads
    - i.e., postpone context switch
  - Others?