Tuning Linux-RT:
Priority Inversion in Workqueue

Jeaho HWANG

저는 이런 사람입니다.

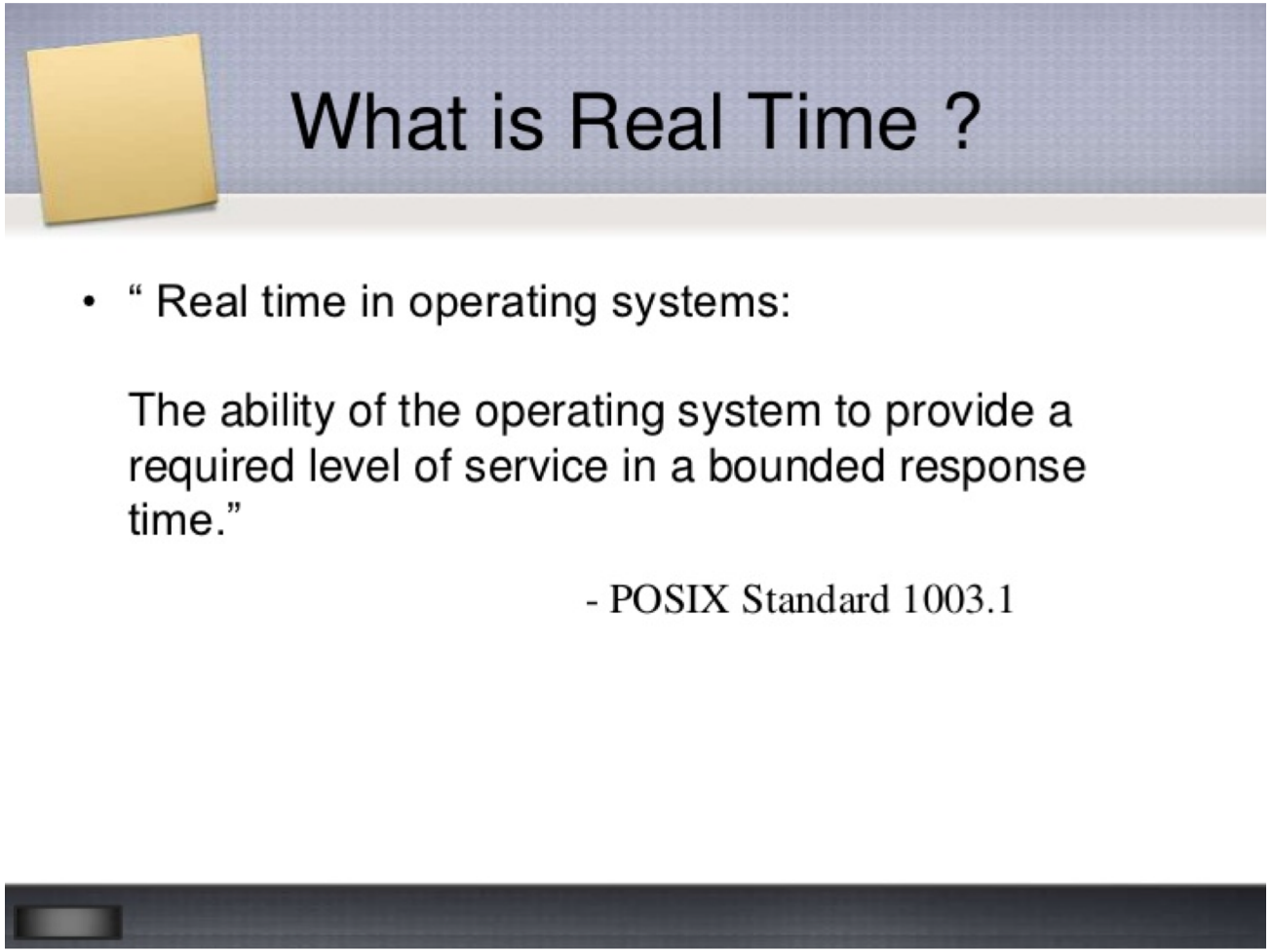
- 황재호
- KAIST 아키랩 Ph.D. 출신
  - 학위는 없어요.
  - 후우...
- RTST 근무중(2014~)
  - "알티스트"라고 읽어주세요.
  - Linux 파트장
  - 대전에 있습니다.
    - 대전 좋아하세요?
- jhhwang@rtst.co.kr

Disclaimer(?)

- 이 발표에서는 아래의 커널 개념들이 서술됩니다.
  - Preempt_rt
  - Ftrace
  - Event tracing
  - Workqueue
  - Priority Inversion
- 시간상 위 개념을 전부 설명하기 힘듭니다.
  - 각 파트에 레퍼런스를 참고해주세요.
  - 전부 잘 알고 계시다면... 알티스트에 오신걸 환영합니다!?!
- **보안상 구체적인 자료 공개가 힘듭니다. 양해 바랍니다.**

Real Time?

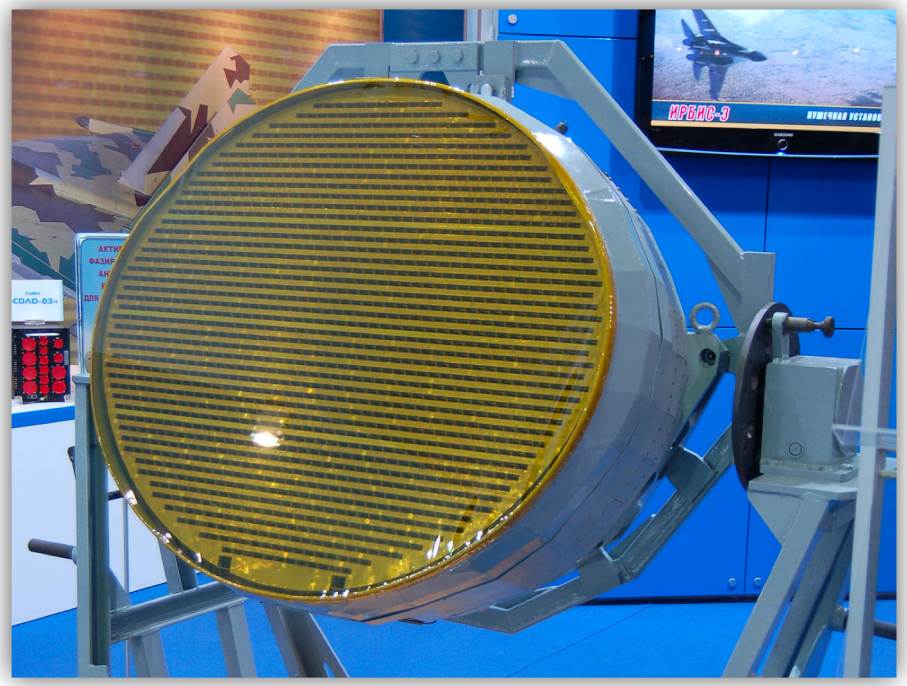
Real Time?


What is Real Time ?
• “ Real time in operating systems:
The ability of the operating system to provide a required level of service in a bounded response time.”
- POSIX Standard 1003.1

Real Time Operating System?

- Processing must be done within the defined constraints **or the system will fail**. [Wikipedia]
  - 주기성
  - 정시성
  - Responsibility >> Throughput
  - Deadline miss -> critical fail

Real Time Operating System?



Real Time Operating System?

The Real Time Linux

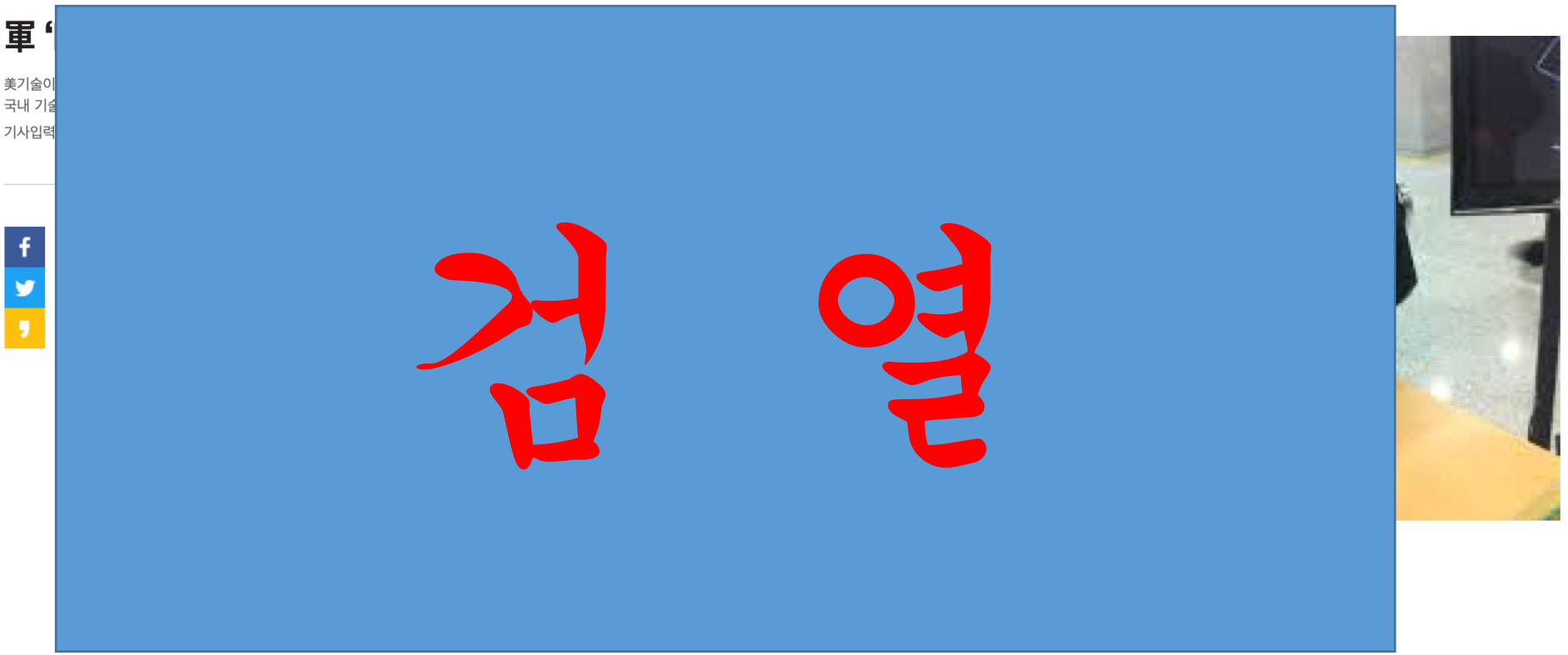
- 많은 real-time target들이 RTOS에서 Linux로 전환중
  - 하드웨어 성능 발달
  - 더 편한 개발 환경 및 library
  - **쌉니다.**

The Real Time Linux

- Preempt-RT
  - 2.6.11부터 적용
  - 5.3-rc 에서 mainlined
  - The primary maintainers of the CONFIG_PREEMPT_RT patch are Ingo Molnar and Thomas Gleixner
    - https://rt.wiki.kernel.org/index.php/Frequently_Asked_Questions
  - spinlock_t and rwlock_t are now preemptible.
  - Implementing priority inheritance for in-kernel spinlocks and semaphores.
  - Converting interrupt handlers into preemptible kernel threads

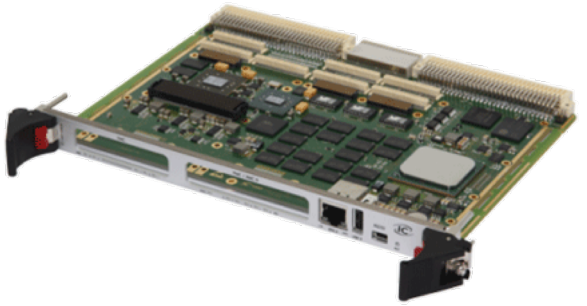
그래서 오늘 할 얘기가...

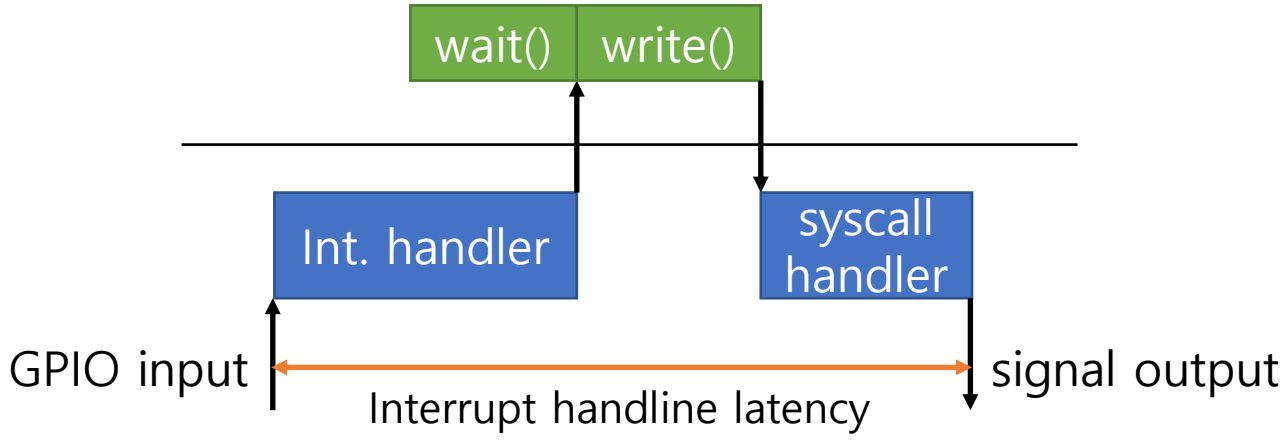
- 대한민국 방위산업을 살린 이야기
  - 나 아니었으면 이거 못 만들었다능...



목표

- 대상에 Linux 적용
  - Xeon 12c 탑재 SBC 타겟
  - GPIO(pca953x)로 1000Hz timing signal 전달
  - Interrupt handling latency < 000us
  - linux-4.19.x-preempt_rt 사용





문제 1: I2C bus가 느려요.

- pca953x GPIO dev는 I2C로 cpu와 통신
  - irq가 발생하면 gpio pin 값을 읽어와 변화 감지를 통해 pending port를 확인
  - Pin read에만 수백us 소모
- We don’t need to know pin value
  - Interrupt signal만 필요
  - 모든 irq event는 pin0에서 발생하는 것으로 간주, read 생략

문제 1: I2C bus가 느려요.

- We don’t need to know pin value
  - **So Happy Ending??**

```
diff -Naur linux-intel-rt-4.19.15-rt12-ic320/drivers/gpio/gpio-pca953x.c linux-intel-rt-4.19.15-rt12-ic320-lowrat/drivers/gpio/gpio-pca953x.c
--- linux-intel-rt-4.19.15-rt12-ic320/drivers/gpio/gpio-pca953x.c   2019-01-17 22:02:39.000000000 +0900
+++ linux-intel-rt-4.19.15-rt12-ic320-lowrat/drivers/gpio/gpio-pca953x.c    2019-07-04 12:54:17.319195497 +0900
@@ -633,8 +640,14 @@
     unsigned nhandled = 0;
     int i;

+#ifdef FAST_INIT
+    /*only first port to be assumed as irq pending, only if any trigger is set*/
+    pending[0] = 0x01;
+    pending[0] &= ( chip->irq_trig_fall[0] | chip->irq_trig_raise[0] );
+#else
     if (!pca953x_irq_pending(chip, pending))
         return IRQ_NONE;
+#endif

     for (i = 0; i < NBANK(chip); i++) {
         while (pending[i]) {
```

그럴리가.....

- 1000Hz 1시간 test 중 1개 꼴로 outlier 발생

Let’s Tune!

- https://www.enea.com/globalassets/downloads/operating-systems/enea-whitepaper---enabling-real-time-in-linux.pdf
- Applying Preempt-RT patch
- Core isolation
- irq isolation
  - Target task가 선점당할 가능성 제거
- CONFIG_NO_HZ
  - 쓸데없는 타이머 이벤트 제거
- 전부 적용했어요.
  - **gpio irq task, target user task 각각 core 독점**

안되잖아?

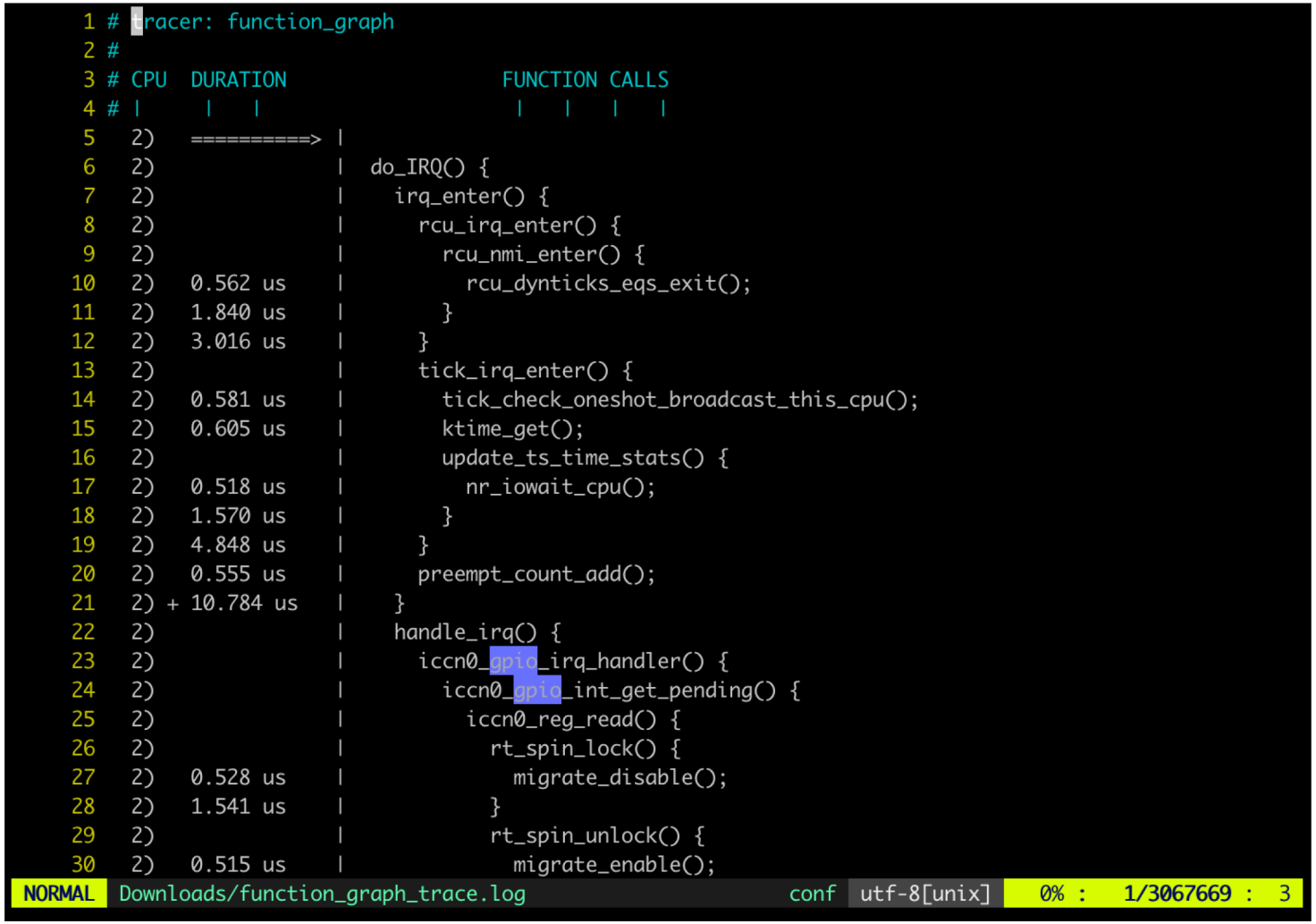
- Average latency는 좋아진듯? - 여전한 outlier
- Hint: no outlier w/o network

Ftrace: 삽질의 시작

- https://www.kernel.org/doc/Documentation/trace/ftrace.txt
- Function_graph로 시작
  - set_graph_function 으로 원하는 function만 graph 획득 가능
  - Function call latency 측정에 큰 도움 안됨 – high cost
  - 호출 경로를 파악하여 지연 요소 확인 가능

Ftrace: 삽질의 시작

- Function_graph로 시작
- 지연 요소 못찾음



Event tracing

- https://www.kernel.org/doc/html/v4.19/trace/events.html
- Function_graph를 통해 interrupt handling 경로 파악
  - Irq handler -> irq thread -> kworker -> user application
  - 각각의 sched_waking, sched_switch 타임스탬프 기록
  - Irq_hadler_entry – app switch time 측정으로 outlier 파악
  - 각 task의 sched latency 확인으로 원인 분석

Event tracing

- Function_graph를 통해 interrupt handling 경로 파악

```
1 $PID_APP=
2 $IRQ_GPIO=
3 $TRACE_PWD=/sys/kernel/debug/tracing
4
5 echo common_pid == $PID_APP > $TRACE_PWD/events/syscalls/sys_exit_select/filter
6 echo next_pid == $PID_APP > $TRACE_PWD/events/sched/sched_switch/filter
7 echo pid == $PID_APP > $TRACE_PWD/events/sched/sched_waking/filter
8 echo irq==$IRQ_GPIO > $TRACE_PWD/events/irq/irq_handler_entry/filter
9 echo irq==$IRQ_GPIO > $TRACE_PWD/events/irq/irq_handler_exit/filter
10 echo 1 > $TRACE_PWD/events/syscalls/sys_exit_select/enable
11 echo 1 > $TRACE_PWD/events/sched/sched_switch/enable
12 echo 1 > $TRACE_PWD/events/sched/sched_waking/enable
13 echo 1 > $TRACE_PWD/events/irq/irq_handler_entry/enable
14 echo 1 > $TRACE_PWD/events/irq/irq_handler_exit/enable
```

Event tracing

- Function_graph를 통해 interrupt handling 경로 파악

```
          <idle>-0     [002] d..h1..  3865.817325: irq_handler_entry: irq=107 name=gpiolib
          <idle>-0     [002] d..h1..  3865.817326: irq_handler_exit: irq=107 ret=handled
          <idle>-0     [011] d...2..  3865.817328: sched_switch: prev_comm=swapper/11 prev_pi
d=0 prev_prio=120 prev_state=R ==> next_comm=irq/107-gpiolib next_pid=2473 next_prio=0
 irq/107-gpiolib-2473  [011] .....11  3865.817332: __wake_up_common <-__wake_up_common_lock
 irq/107-gpiolib-2473  [011] d...2..  3865.817333: sched_switch: prev_comm=irq/107-gpiolib pr
ev_pid=2473 prev_prio=0 prev_state=S ==> next_comm=kworker/11:1 next_pid=121 next_prio=49
    kworker/11:1-121   [011] .....12  3865.817335: __wake_up_common <-__wake_up_common_lock
    kworker/11:1-121   [011] d...112  3865.817335: sched_waking: comm=clihdlc-tst-gpi pid=258
6 prio=0 target_cpu=010
          <idle>-0     [010] d...2..  3865.817337: sched_switch: prev_comm=swapper/10 prev_pi
d=0 prev_prio=120 prev_state=R ==> next_comm=clihdlc-tst-gpi next_pid=2586 next_prio=0
```

Event tracing

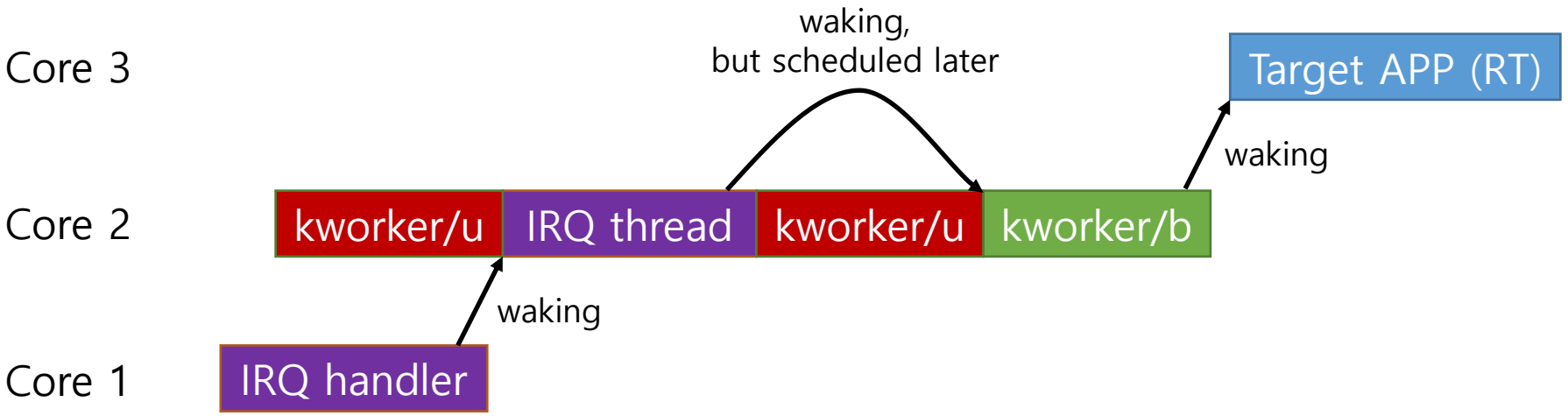
- https://www.kernel.org/doc/html/v4.19/trace/events.html
- Function_graph를 통해 interrupt handling 경로 파악
  - Irq handler -> irq thread -> kworker -> user application
  - 각각의 sched_waking, sched_switch 타임스탬프 기록
  - Irq_hadler_entry – app switch time 측정으로 outlier 파악
  - 각 task의 sched latency 확인으로 원인 분석
- **kworker sched에서 outlier 원인 발견**

Workqueue: the source of PI

- http://jake.dothome.co.kr/workqueue-1/
  - Interrupt bottom half 처리를 위한 매커니즘
- Bound/unbound workqueue로 분류됨
  - Bound workqueue – 실행될 cpu 및 thread가 정해져 있음
  - Unbound workqueue – 그때그때 thread를 생성해 임의의 CPU 사용
  - GPIO waiting app – RT prio, sched from bound worker
  - Network job – non-RT, done from unbound worker
- Kworker는 전부 non-RT, 따라서 둘은 preempt되지 않음
  - **Unbound kworker가 실행중인 경우 bound kworker sched 지연**
  - **Priority Inversion!!**

Workqueue: the source of PI

- Kworker는 전부 non-RT, 따라서 둘은 preempt되지 않음
  - **Unbound kworker가 실행중인 경우 bound kworker sched 지연**
  - **Priority Inversion!!**



여태 왜 몰랐죠?

- Linux is a GENERAL PURPOSE OS
  - 철저한 추상화 – gpio에 대한 generalized fops 지원
  - Throughput >> latency
  - I/O에서 00us 짜리 jitter를 누가 신경써요? – 저희요.
  - I/O latency optimization에 대한 많인 수요와 이슈 예상
    - 빠른 device 등장
    - Real time 수요 증가

어떻게 해야 잘 해결했다고 소문이 날까?

- /sys/devices/virtual/workqueue/cpumask - 실패
  - Workqueue 실행 affinity 조정
  - Bound workqueue도 함께 제외됨
- Proof of Concept
  - Irq thread 독점 core – workqeue도 배당 – 의 kworker에 RT prio 부여
    - 현재 case에서는 정상동작 – gpio가 코어 및 해당 worker 독점
    - 글쎄… 별로 좋은 해결 방법은 아닌거 같죠?
- 독자 API와 lib 사용
  - 일부 보드에서 사용중
    - select/poll/read/write 대신 ioctl을 사용한 독자 lib 사용
    - 자체 wait queue 사용 – irq thread에서 바로 wakeup

어떻게 해야 잘 해결했다고 소문이 날까?

- 어떻게든 workqueue priority를 높여보자
- system_highpri_wq 사용 (nice -20)
  - sched_work 호출은 gpio generic code – work semantic 모름
  - Priority Inherit를 위한 추가 구현 필요
    - 현재 kernel concept 위에서 가능할지?
  - 같은 non-rt면 preemption 보장 없음 – CFS
    - System_rtpri_wq 구현 필요성?

- **Any Better Idea?**
  - **Do you want to contribute to kernel with me?**

결론: system enginering은 삽질의 연속

- Real Time은 특히나 튜닝 이슈
  - Hard to find general solution
  - Resource, driver tuning
  - 예외를 최대한 없애자!

- RTST는 이런 거 하는 회사입니다.
  - Linux, XEN, KVM 등 core를 만지고 싶으신 분
  - OS를 직접 만들어보고 싶으신 분
  - Linux/RTOS 시스템 삽질에 소질 있으신 분
  - 소중히 모십니다. 어서오세요. 감사합니다 ☺

RTst
Reliable & Trustworthy