Realtime Application 성능 최적화를 위한 RT-Linux 환경 구성

RTST Linux Team
황재호
jhhwang@rtst.co.kr

About this talk...

- Linux에서 real time 환경을 구축할 때 고려해야 할 것들 이야기
  - 5.x 커널(PREEMP_RT)을 대상으로 한 경험 위주
    - PREEMPT_RT 커널을 사용
  - 커널 코드 자체를 분석하거나 수정하지 않음
  - C를 사용한 application 개발을 가정
    - Modern C++ 잘 모릅니다 ㅠ
    - Rust? with libc crate?

PREEMPT_RT

- RT task는 deadline miss를 피하는게 최우선
- 이를 위해선 Determinism 확보가 필요 -> latency 최소화 필요
- Latency 최소화를 위해: 선점 불가능 구간의 최소화
- **PREEMPT_RT: fully preemptible kernel**
  - Spin lock -> sleeping mutex
  - Thread irq handler
  - Priority inheritance
  - local_bh_disable 제거(하는중)

Realtime 성능 향상 기술

- **Scheduler selection**
- Timer, Thread
- **CPU, Memory**
- **Synchronization**
- **I/O**
- **Kernel Configurations**

RT consideraton – Scheduler

- RT scheduler 사용
  - SCHED_FIFO, SCHED_RR, SCHED_DEADLINE 등
    - sched_setscheduler()
  - CFS, EEVDF는 우선순위를 보장하지 않음
  - **Priority 99는 사용하지 말 것(watchdog 등 최우선 kernel thread 전용)**
    - RT task의 CPU 독점을 막을 것

RT consideraton – CPU

- CPU affinity
  - 태스크를 특정 코어에 고정적으로 할당함으로써 determinism을 높일 수 있다.
    - Taskset, cgroups, isolcpus 등 사용
- IRQ Affinity
  - RT task들의 core에서 irq handler로 인한 지연을 방지
  - /proc/irq/[irq-number]/smp_affinity
  - Workqueue 역시 affinity 지정 가능 (echo 1 > /sys/devices/virtual/workqueue/cpumask)
- NO_HZ_FULL kernel configuration
  - 주기적 tick으로 인한 지연 방지
  - rcu_nocbs 포함 – RCU callback이 다른 CPU로 offloading 됨
- Cpufreq performance governor 사용
  - CPU state change로 latency 증가

RT consideraton – Memory

- Page fault로 인한 jitter 방지
  - **mlockall() 사용**
  - startup time에 모든 스레드 생성
  - Memory pool을 이용한 malloc/free – 가능하면 dynamic alloc 자제
  - Pagefault 유발 syscall 사용에 유의 – mmap 등
  - 전역변수는 RT 성능에 영향을 주지 않음
  - Non-RT task의 mem op가 RT task 성능에 영향을 끼칠 수도…

RT consideraton – Synchronization

- RT mutex 사용 - Priority inheritance 보장

```
pthread_mutexattr_setprotocol(attr, PTHREAD_PRIO_INHERIT);
pthread_mutex_init(mut, attr);
```

- Spinlock 사용금지!
  - 애초에 스핀락을 안쓰는게 preempt-rt 커널임
  - RT priority로 sleep 없는 무한루프 사용금지
- 프로세스간 데이터 공유 – POSIX IPC 사용
  - Conditional variable은 priority를 보장하지 않음
- Signal 사용은 지양

RT consideraton – I/O

- 웬만하면 쓰지 말것
  - Determinism의 주적
  - 비동기를 사용하거나 non-RT task를 통할 것
  - **VGA 콘솔 절대 사용금지**
    - 매우 긴 latency 가능성
    - 모니터 wakeup 이벤트가 최악

Kernel Configuration Optimization

- RT에 도움이 되는 기능
  - NO_HZ_FULL (Timer interrupt 횟수를 줄여 jitter와 시스템 overhead를 줄임)
  - RCU_NOCB_CPU (NO_HZ_FULL에 의해 자동 enable)
- 커널 정보 수집, statistics, 디버깅 관련 기능들: 런타임에 정보들을 기록하는 오버헤드 감소
  - IRQ_TIME_ACCOUNTING, BSD_PROCESS_ACCT, TASKSTATS, PSI (CPU/Task time and stats accounting)
  - CPU_FREQ_STAT (CPU frequency governor의 상태 변경에 대한 정보 기록)
  - X86_CPA_STATSTICS (huge page mapping 관련 메커니즘을 도와줄 수 있는 정보 수집용)
  - IDLE_PAGE_TRACKING (일정기간동안 접근되지 않은 페이지 추적)
  - PERCPU_STATS
  - VM_EVENT_COUNTERS
  - DEBUG_* (각종 debug용 정보 수집, 메시지 출력 기능들)

Kernel Configuration Optimization

- 기타 스케줄링 features: 스케줄링 오버헤드 감소
  - SCHED_CORE (SMT sibling 고려 task-core assign)
  - UCLAMP_TASK(특정 태스크의 min-max utilization 설정을 위한 Utilization clamping 기능)
  - SCHED_AUTOGROUP (자동으로 일부 태스크들을 그룹으로 묶어 스케줄링)
  - SCHED_MC/SCHED_CLUSTER/SCHED_DEBUG/SCHEDSTATS
- 보안
  - STACKPROTECTOR, STACKPROTECTOR_STRONG
- 그 외 사용하지 않는 기능들:
  - CHECKPOINT_RESTORE
  - X86_5LEVEL(5 Level page table)
  - MEMORY_HOTPLUG
  - HYPERVISOR_GUEST(Guest VM으로 사용되지 않을거라면)
  - Deprecated, Old, Obsolete, Experimental 기능들
  - ex. SYSFS_DEPRECATED

RT consideraton – misc

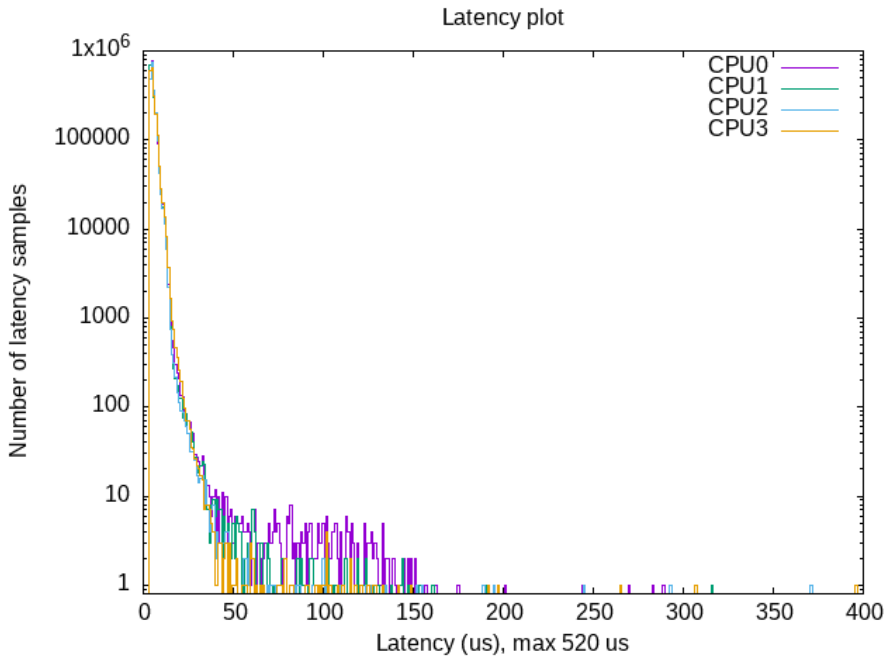
- RT throttling
  - RT task로 인해 CPU가 lockup 되는것을 막기 위해 runtime을 소모하면 task 차단
  - 이로 인해 뜻하지 않은 jitter 발생 가능성; runtime -1 설정으로 disable 가능
- clock_nanosleep; TIMER_ABSTIME, CLOCK_MONOTONIC 사용
- 코어당 task는 적을수록 좋음
  - 필요시 user thread, user level batch 등 사용 가능
- Vmstat 갱신주기 늘리기

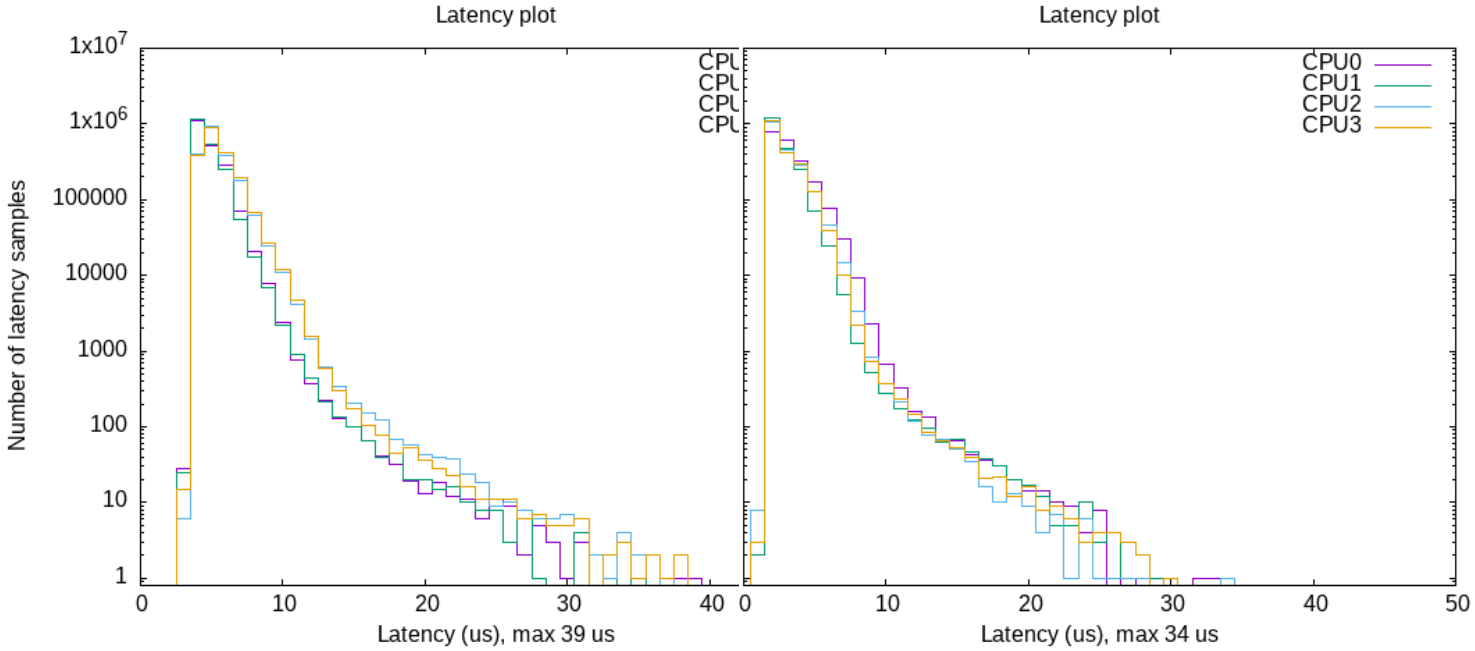
RT Performance Evaluation - cyclictest

- OSADL (Open Source Automation Development Lab)의 test command
  - cyclictest -l100000000 -m -Sp90 -i201 -h400 -q >output
    - 201us 주기로 1억번(2000초 = 5.56시간) 수행
      - Lockstep sampling 방지
    - (switched time – wakeup time) 으로 schedule latency 측정
    - Min, average 값보다 max latency가 RT 성능에 중요
- 테스트 방식
  - Background stress로 memcpy 사용

RT Performance Evaluation - cyclictest

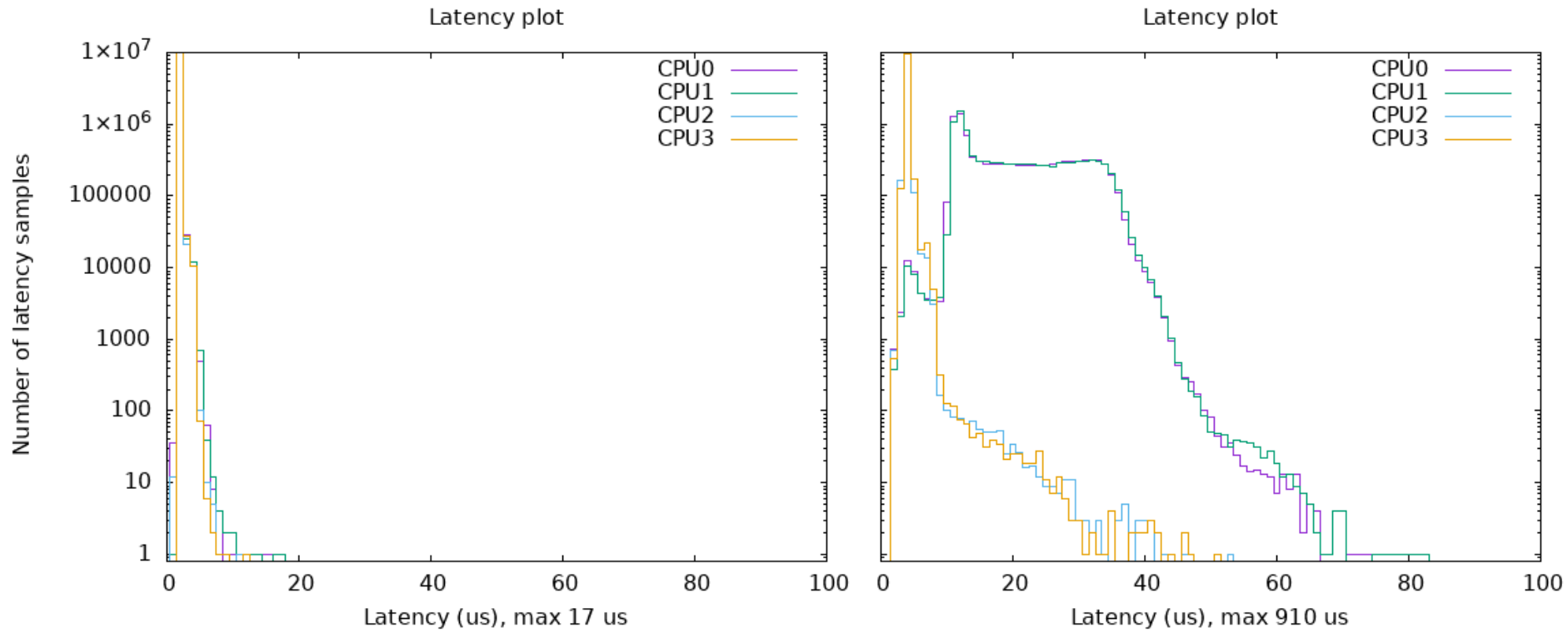
- No RT Vs. RT Vs. optimized





RT Performance Evaluation - cyclictest

- isolcpus & mem fault effects
  - CPU 2,3 isolated w/ stress-ng userfault



감사합니다!

The best embedded system software company in Korea

Vision Make the World Safer and More Secure

Value Reliable & Trustworthy

RTst Reliable & Trustworthy

RTWORKS
Real-time OS

RTST Linux
Linux

RTST AI
맞춤형 Edge AI

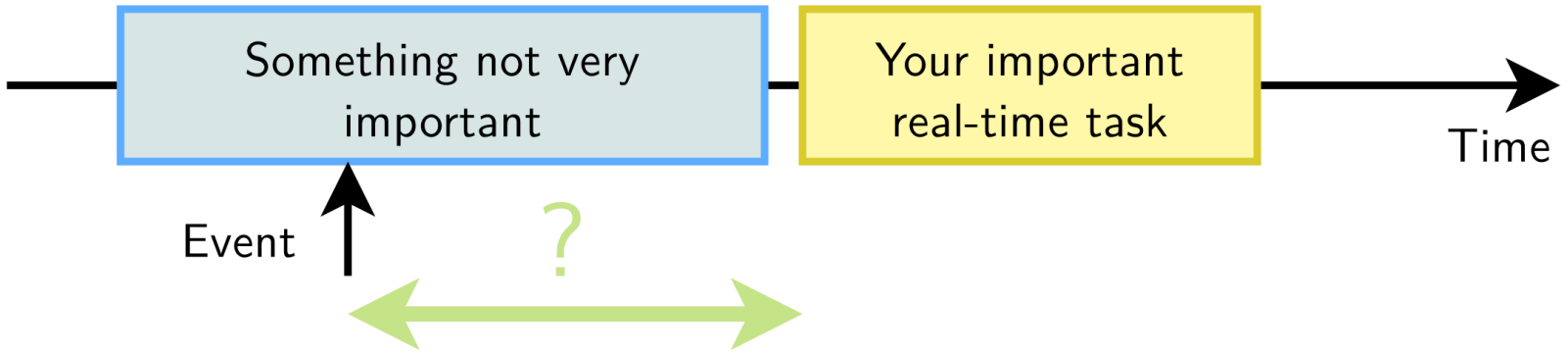
Determinism

- 멀티태스킹 시스템에서 중요한 작업은 deterministic 해야 함
- CPU 공유 및 외부 인터럽트의 영향을 완전히 예측할 수 있어야 함
- 디자인 제약:
  - 예측 불가한 상황을 최대한 회피 해야 함
  - sleep을 길게 하거나 너무 빠르게 일어나는 것도 피해야 함
  - 시스템이 fully preemptible 해야 함
  - 시스템의 최대한 많은 부분을 컨트롤 하도록 해야 함

Latencies

지연시간: 이벤트와 이벤트에 대한 반응 사이의 경과 시간

- 최악의 경우 실행 시간은 예측하기 매우 어려움
- 최악의 경우의 반응 시간에 대한 제한이 필요함
- **실시간 운영 체제에서는 지연 시간이 핵심임**



PREEMPT_RT

- multi-task 실시간 운영 체제를 구현하는 한 가지 방법은 preemptible system을 갖추는 것임
  - 언제든지, 어떤 작업이라도 중단하여 높은 우선순위의 작업을 실행할 수 있어야함
- 리눅스에는 이미 사용자 공간 preemption이 지원함
- 리눅스 커널은 실시간 스케줄링 정책도 지원함
- 그러나 커널 모드에서 실행되는 코드가 전부 preemption될 수는 없음
- PREEMPT_RT 패치는 **커널 모드에서 실행되는 모든 코드를 preemptible하게 만드는 것**이 목표임
  - Spin lock -> sleeping mutex
  - Interrupt handler -> threaded irq
  - PI mutex

Why RT Patch?

why use PREEMPT_RT:

- POSIX/Linux API => 간편함
- 대부분의 하드웨어에서 사용 가능함
- 일반적인 응용과 함께 RT 작업을 돌릴 수 있음
- 커뮤니티의 도움이 용이

why not to use PREEMPT_RT:

- 정석 RTOS가 아님
- 리눅스가 지원하는 일반적인 하드웨어는 RT를 고려하지 않음
- RT패치는 최고의 성능을 위한 것은 아님
- **여전히 tuning이 필요**