<Agenda>

CE Linux Forum

Korea Tech Conference

2005년 5월 14일, 서울

CE Linux Forum

<Agenda>

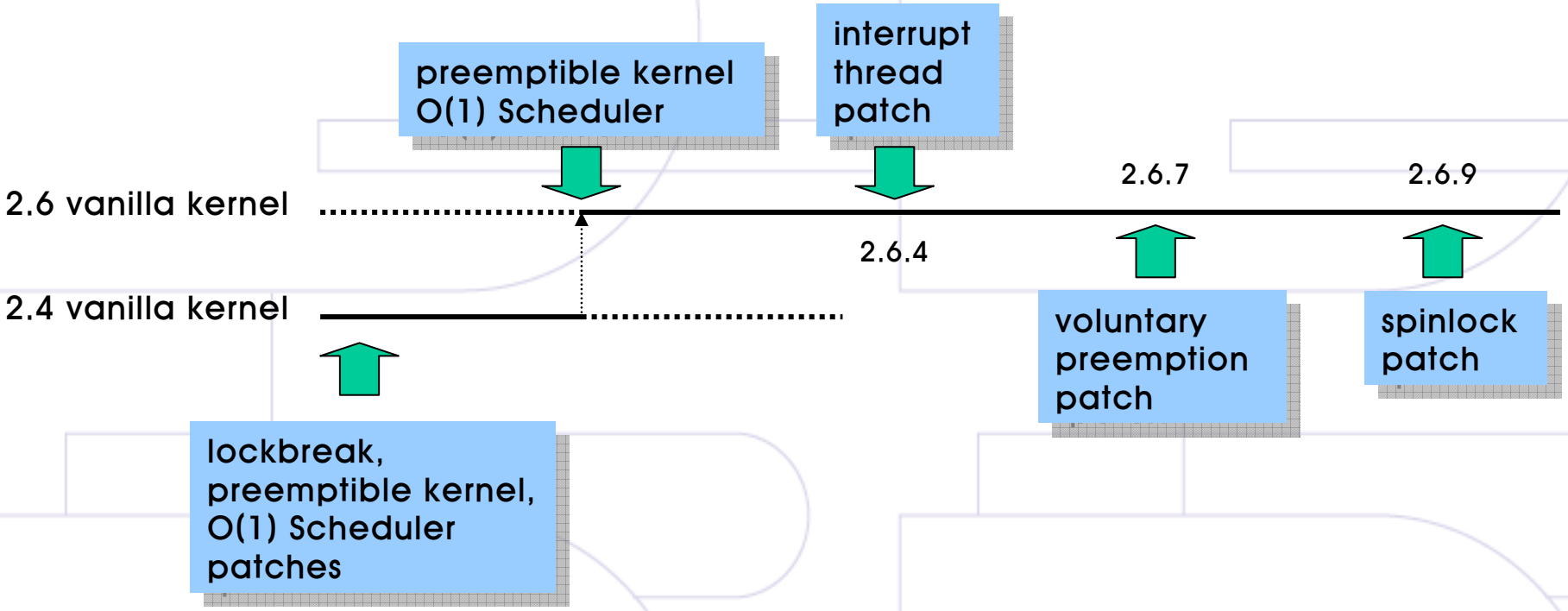
리눅스에서의 실시간 지원

정영준 / 임용관

목 차

개 요

- Throughput vs Realtime
- 실시간 관련 커널 패치



<Agenda>

Welcome - Mozilla Firefox

http://www.celinuxforum.org/

CE Linux Forum

Welcome

Welcome to the CE Linux Forum web site.

PatchArchive - CE Linux Public - Mozilla Firefox

http://tree.celinuxforum.org/CelfPubWiki/PatchArchive

PatchArchive

CE Linux Patch Archive

Table Of Contents:

1. CE Linux Patch Archive
   1. Patches for 2.6.11
   2. Patches for 2.6.10
   3. Patches for 2.6.9
   4. Patches for 2.6.8.1
   5. Patches for 2.6.7
   6. Patches for 2.6.6
2. Patch Table Key
3. Instructions to Download and Apply

Patches for 2.6.11

Name	Patch File	Base Kernel	Status	Patch Home Page	Comments
celf-pm-patches-interface.tar.bz2	celf-pm-patches-interface.tar.bz2	linux-2.6.11	experimental	.	Interface Magazine article.

Patches for 2.6.10

Name	Patch File	Base Kernel	Status	Patch Home Page	Comments
Linux Trace Toolkit 0.9.6 kernel patches	ltt-2.6.10-tb5.tar.gz	linux-2.6.10	experimental	LinuxTraceToolkit	.
RTC no-sync, for PPC	rtcnosync-ppc-2.6.10.patch	linux-2.6.10	tested - ppc:OK	RTCNoSync	.
printk-times	printk-times-2.6.10.patch	linux-2.6.10	tested - ppc:OK	PrintkTimes	.
RBTX49xx platform support	rbtx49.linux-2.6.10.patch	linux-2.6.10	tested - mips:OK	RBTX49xxPlatformSupport	.

Patches for 2.6.9

Name	Patch File	Base Kernel	Status	Patch Home Page	Comments
Linear Cramfs support v3	cramfs-linear-xip-3.patch	linux-2.6.9	experimental	ApplicationXIP	.
Linear Cramfs support v2	cramfs-linear-xip-2.patch	linux-2.6.9	experimental	ApplicationXIP	.
Linear Cramfs	cramfs-linear-xip.patch	linux-2.6.9	experimental	ApplicationXIP	.

스케줄러

2.4 커널의 스케줄러

- Throughput 기반의 단일 runqueue 지원
- 실시간 태스크 지원 미흡
  - 실시간 태스크와 일반 태스크가 runqueue에 혼재
  - Non-deterministic mechanism
    - 태스크들의 수에 따라서 고정된 시간내에 스케줄링이 일어날 수 없음
  - 실시간 성능이 요구되는 시스템에는 적합하지 않음
  - 실시간 태스크들에 한해서는 제한된 시간내의 스케줄링 지연시간 지원이 요구됨
    - O(1) 스케줄링

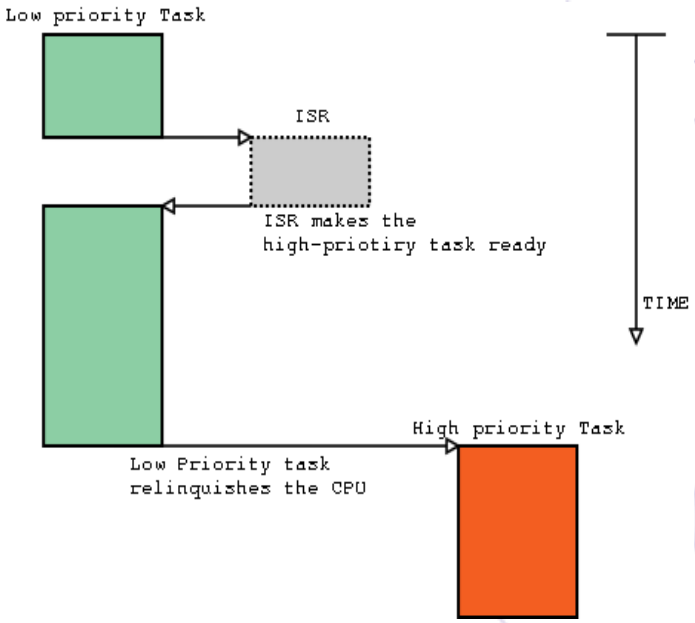
2.6 커널의 스케줄러

- 우선순위 기반 두개의 runqueue
- 실시간성 지원
  - runqueue에 우선순위별로 태스크들이 연결됨
  - 비트맵에서 처음으로 1인 비트를 찾는 연산만으로 다음에 실행될 태스크를 선택할 수 있음 – bsfl
  - deterministic mechanism
    - 태스크들의 수가 아무리 많다 하여도 고정된 시간내에 스케줄링이 일어남
  - 실시간 성능이 요구되는 시스템에 적합

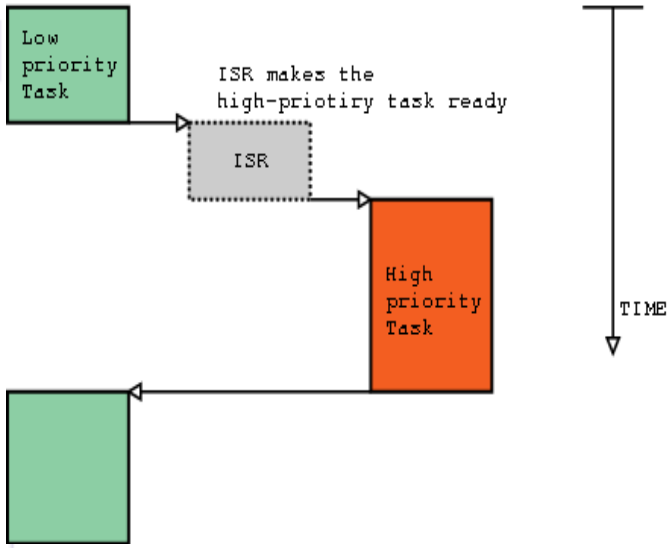
커널 구조

선점형 커널 지원

- 커널 모드 선점 비교 (차이점)



<비선점형 커널>



<선점형 커널>

선점형 커널 지원

- 개요
  - 커널 2.4.x에서부터 오픈 커뮤니티에서 시작
  - 커널 2.5.4에서 공식 채택
  - 커널 2.6.x에서 컴파일시에 옵션으로 제공
  - Concurrency & Reentrancy 지원
  - SMP locking mechanism 수정 지원
    - 이유와 고려사항
      - 선점형 커널을 위한 새로운 락메커니즘 지원 필요
      - 선점형 커널의 concurrency와 reentrancy 문제는 SMP에서의 그것과 같음
      - 기존 락메커니즘 Semantic을 유지하면서 SMP locking mechanism을 수정
    - 장점
      - 최소한의 커널 수정으로 preemptible kernel 구현
  - 결과
    - Preemptible kernel 지원
      - 시스템 responsibility 증가, 시스템 throughput 감소

선점형 커널 구현

- 새로운 Preemption lock 필요
  - 선점형 커널 : “Kernel은 preemption locked region을 제외하고 preemptible 하다”
  - 효과적인 락메커니즘 수정
    - SMP spinlock

```
/*preemption enable region */
preempt_lock();
original_spin_lock();        → new_spin_lock();
/* preemption locked region */
original_spin_unlock();      → new_spin_unlock();
preempt_unlock();
 /*preemption enable region */
```

    - 락메커니즘의 Semantic은 유지
    - 락메커니즘 자체만을 수정하여 대부분의 커널 코드가 선점 가능함

선점형 커널 구현

- Preemption lock 의 추가적인 사용
  - SMP-safe vs. Preempt-safe
  - SMP Spinlock 이 커버되지 않는 부분은 Preemption lock을 사용해야 함
    - Preempt-safe 로 보호해야하는 곳
      - Per-CPU data structures
      - CPU state : this is very architecture dependent
      - 예제 코드

```
cpucache_t *cc; /* this is per-CPU */
        preempt_disable();
        cc = cc_data(searchp);
        if (cc && cc->avail) {
                __free_block(searchp, cc_entry(cc), cc->avail);
                cc->avail = 0;
        }
        preempt_enable();
        return 0;
```

상세 구현 들여다보기

- 관련 자료구조
  - TIF_NEED_RESCHED, preempt_count
- Preemption lock

```
#define preempt_disable() do { inc_preempt_count(); barrier(); } while (0)
#define preempt_enable() do { barrier(); dec_preempt_count(); preempt_check_resched(); } while (0)
```

- Spinlock

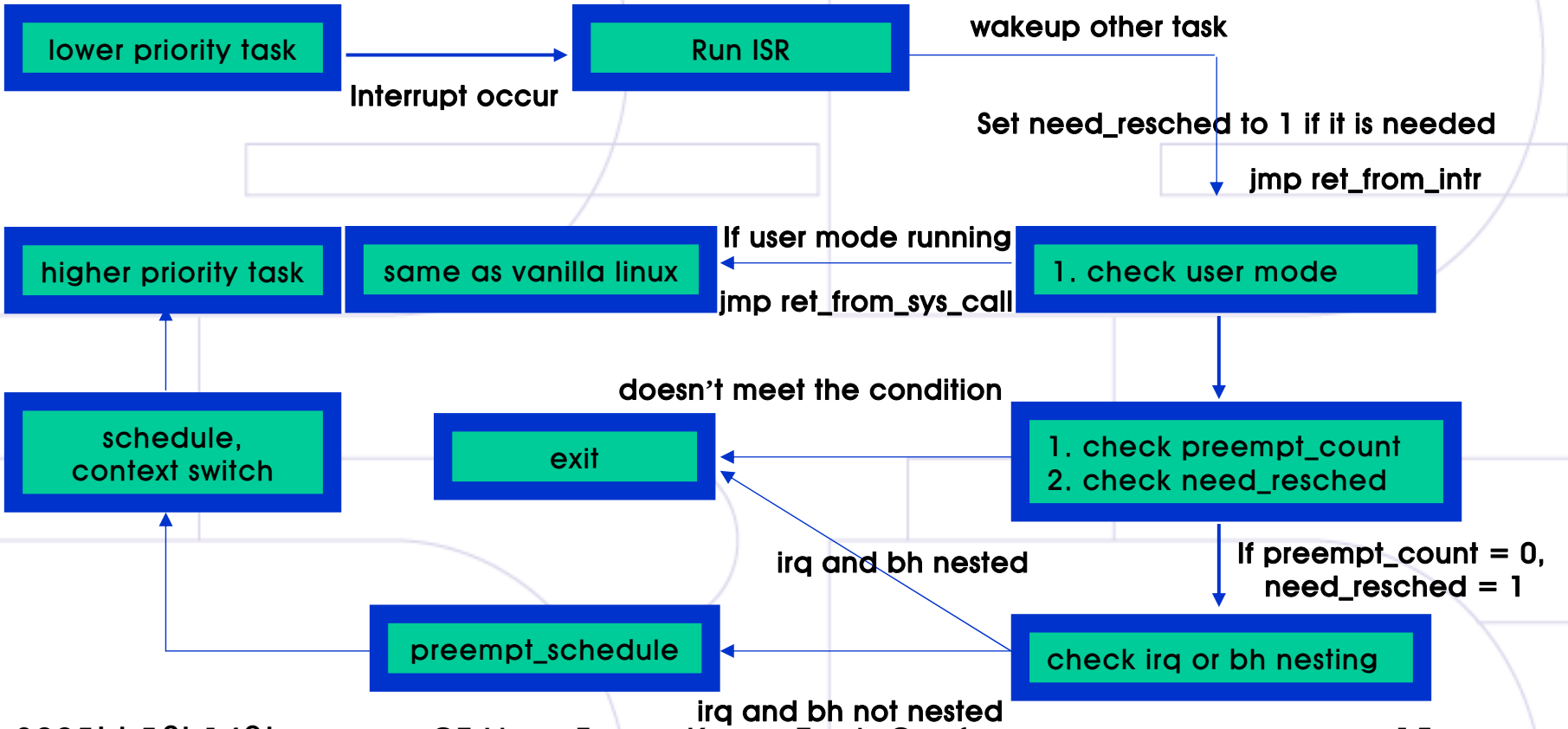
```
#define spin_lock(lock) do { preempt_disable(); _raw_spin_lock(lock); } while(0)
#define spin_trylock(lock) ({preempt_disable(); _raw_spin_trylock(lock) ? 1 : ({preempt_enable(); 0;});})
#define spin_unlock(lock) do { _raw_spin_unlock(lock); preempt_enable(); } while (0)
```

- Interrupt

```
#define spin_lock_irq() do { local_irq_disable(); preempt_disable(); _raw_spin_lock_flags(); } while (0)
#define spin_unlock_irq() do { _raw_spin_unlock(); local_irq_enable(); preempt_enable(); } while (0)
#define spin_lock_bh() do { local_bh_disable(); preempt_disable(); _raw_spin_lock(); } while (0)
#define spin_unlock_bh() do { _raw_spin_unlock(); preempt_enable(); _local_bh_enable(); } while (0)
```

어떻게 적용되어 돌아갈까?

- 인터럽트에서 복귀시에 조건을 만족하면 preempt_schedule을 호출하여 재스케줄링



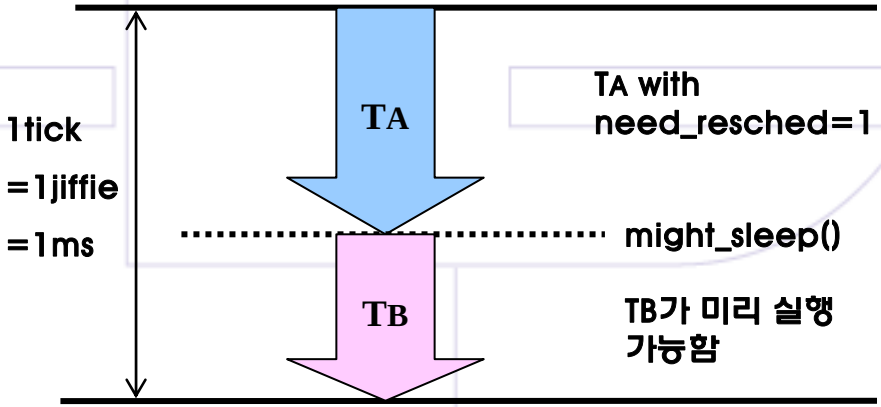
자발적 선점 패치

- 개요
  - 높은 스케줄링 레이턴시
    - reported by JACKit (up to 50ms!)
  - 소스 코드에 새로운 스케줄링 포인트를 추가
    - might_sleep()
      - lock을 갖지 않았으므로 sleep될 수 있음을 나타내는 코드
      - noop이나 디버깅 옵션을 넣으면 lock을 가지고 휴면하게 되는 경우인지를 체크할 수 있음
    - 1ms이상의 레이턴시를 가지는 구간을 탐색 및 테스트
      - 선점 포인트를 삽입
  - 선점 타이밍을 앞당김으로 우선순위가 높은 태스크가 실행될 수 있는 시간을 앞당김
  - 정규 커널 트리에는 미포함
  - 실시간 패치에 통합되어 배포

새로운 스케줄링 포인트

```
# define might_sleep() do { voluntary_resched(); } while (0)

int __sched voluntary_resched(void)
{
#ifdef CONFIG_DEBUG_SPINLOCK_SLEEP
        __might_sleep(__FILE__, __LINE__);
#endif
        if (!voluntary_preemption)
                return 0;
        if (need_resched() && system_state >=
                        SYSTEM_BOOTING_SCHEDULER_OK) {
                set_current_state(TASK_RUNNING);
                schedule();
                return 1;
        }
        return 0;
}
```

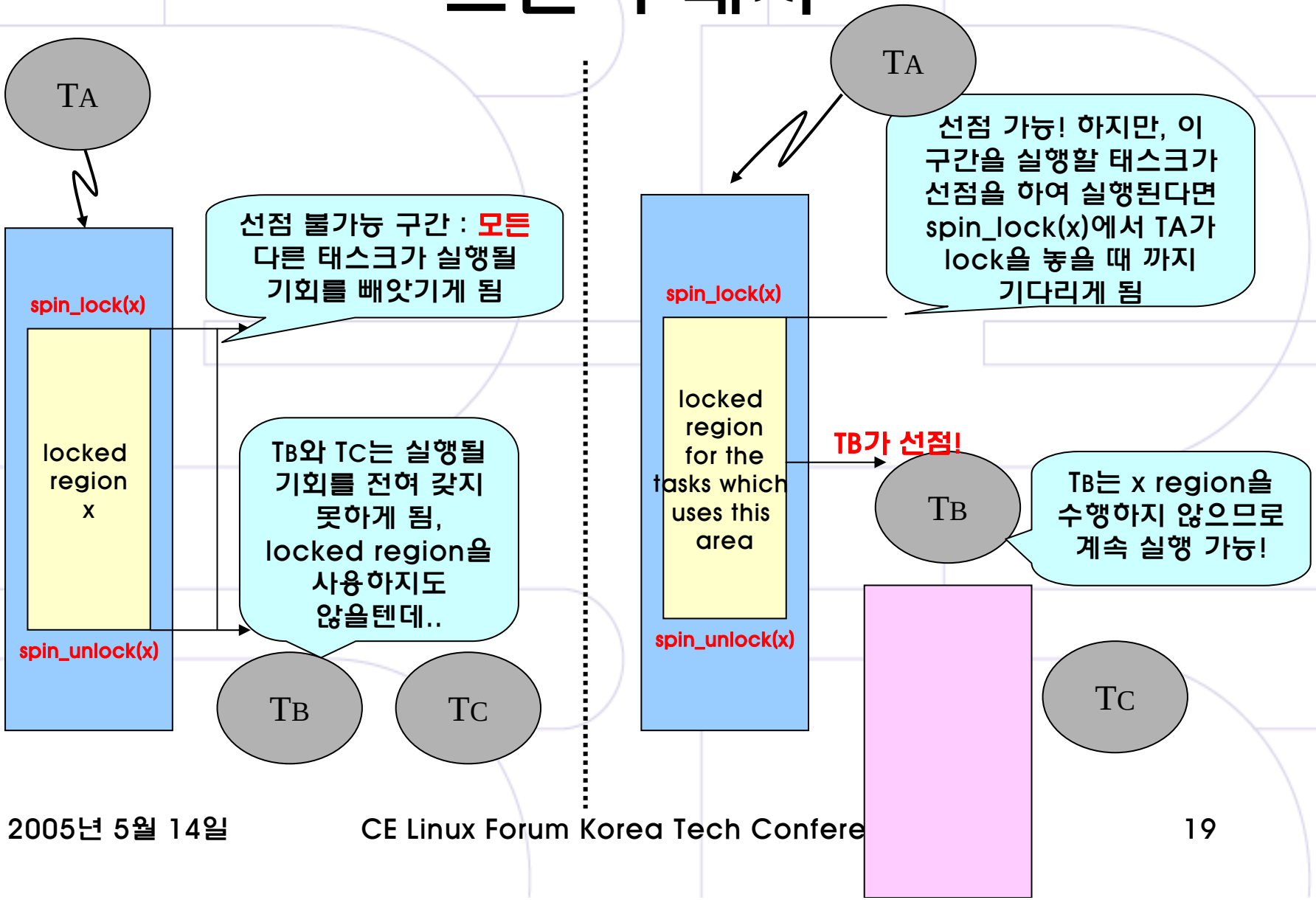


스핀락 패치

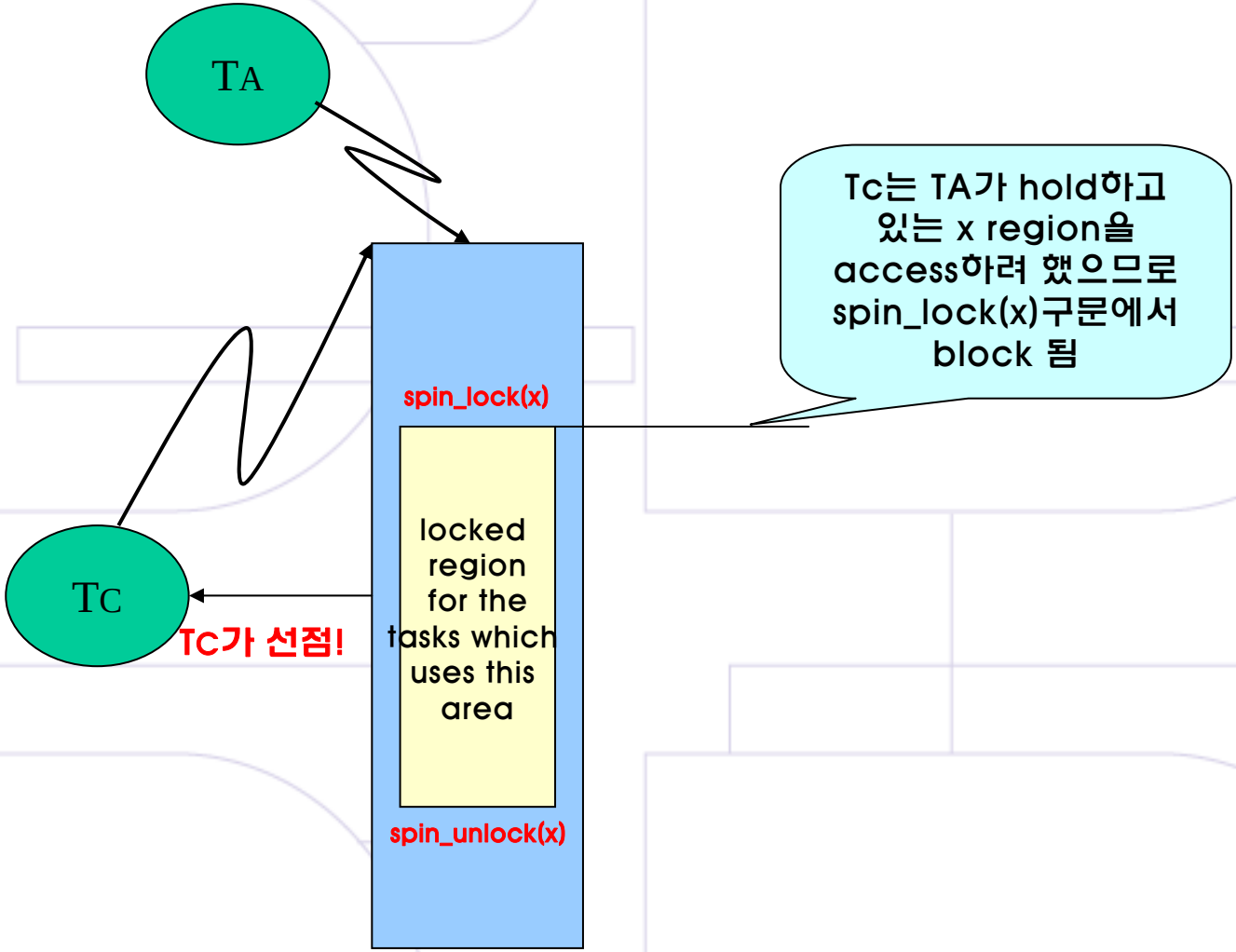
- 선점의 관점에서 본 스핀락
  - 스핀락 구간은 preemption disable 구간임
  - lock을 사용하지도 않는 태스크가 실행될 기회를 빼앗기게 됨
  - 어떤 lock이 잡혀있다면 그 lock을 사용할 태스크만 실행되지 못하게 만드는 것이 필요
- 새로운 lock 매커니즘
  - 90여개의 spin_lock은 오리지널 코드를 유지
  - 나머지 코드는 mutex_lock으로 대체

<Agenda>

스핀락 패치



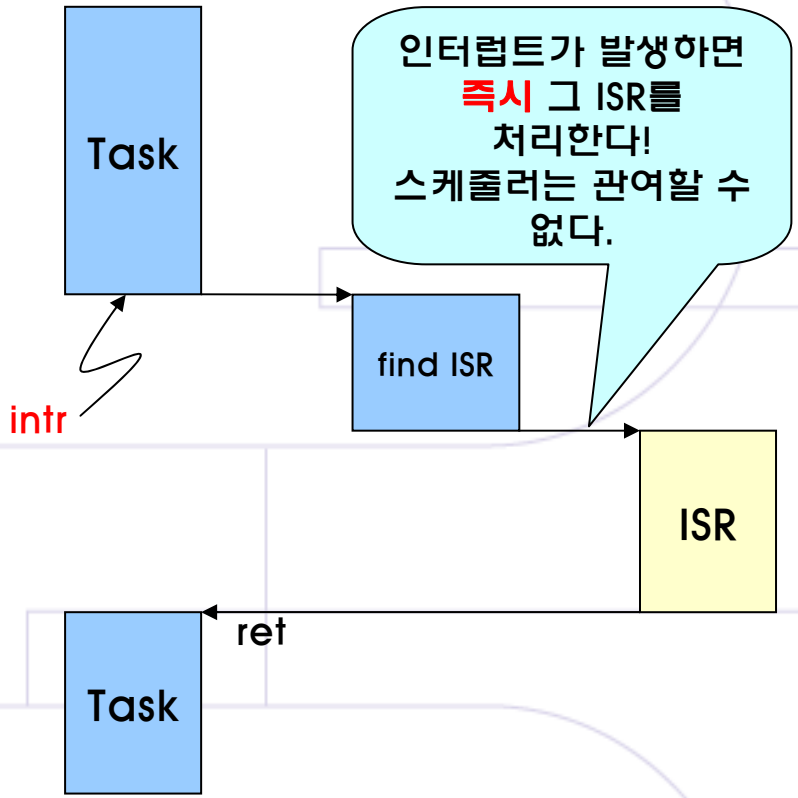
스핀락 패치


TA
Tc는 TA가 hold하고 있는 x region을 access하려 했으므로 spin_lock(x)구문에서 block 됨
spin_lock(x)
locked region for the tasks which uses this area
spin_unlock(x)
TC
Tc가 선점!

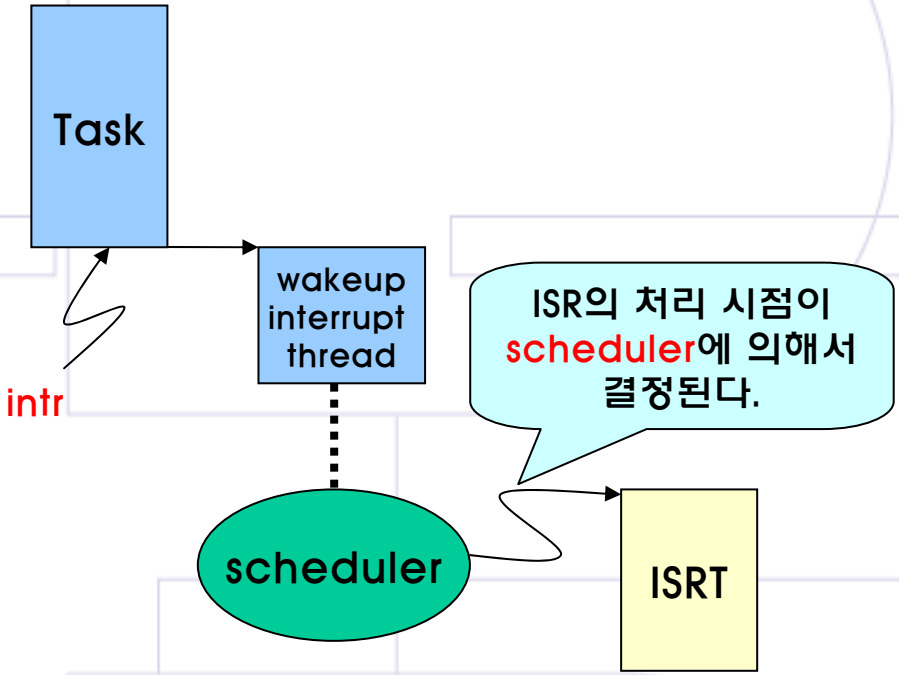
인터럽트 패치

- 개요
  - 2.6.4용 인터럽트 쓰래드 패치 첫 등장
  - 타이머 인터럽트를 제외한 모든 인터럽트를 쓰레드화
    - serialize된 인터럽트 처리를 우선순위화
  - 각 인터럽트 쓰레드에게 실시간 우선순위와 SCHED_FIFO 스케줄링 policy를 할당
  - 스케줄링을 통한 예측가능성

인터럽트 패치



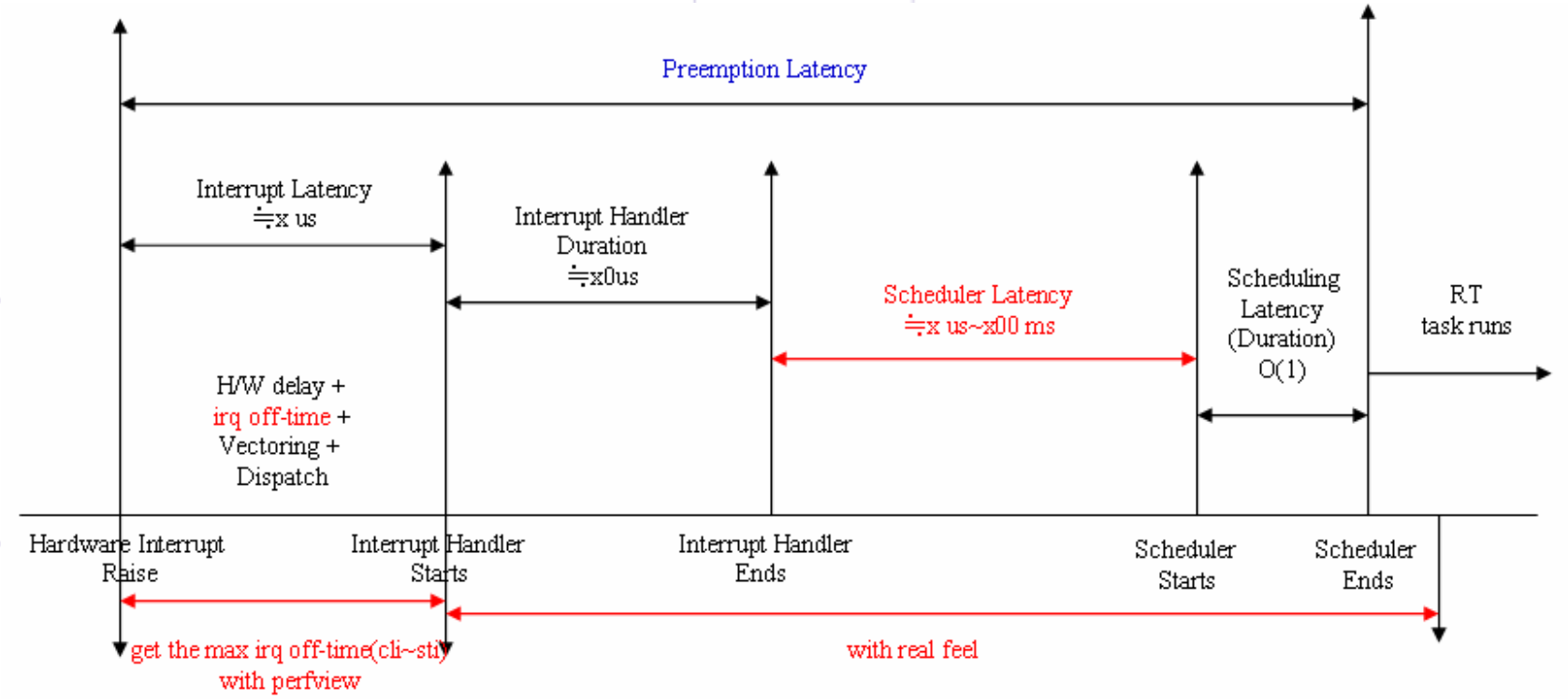
< 기존의 인터럽트 처리 과정 >



< 인터럽트 쓰레드화 >

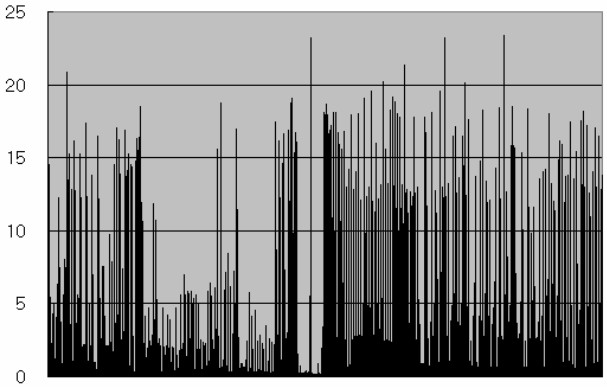
성능 분석

- 인터럽트 관련 시스템 성능

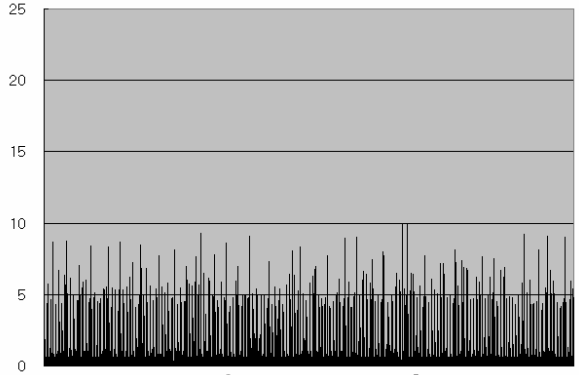


성능 분석

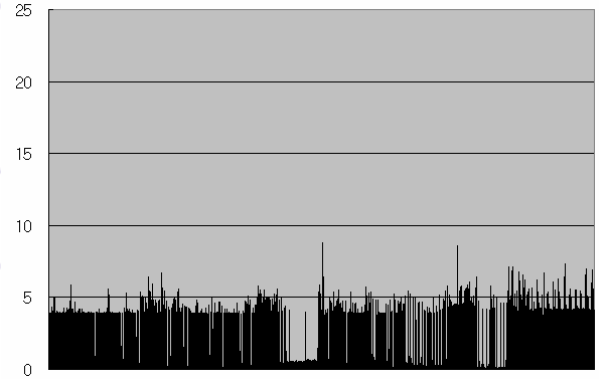
- 인터럽트 지연시간 측정



< 2.6.9-vanilla >



< 2.6.9-인터럽트패치 >

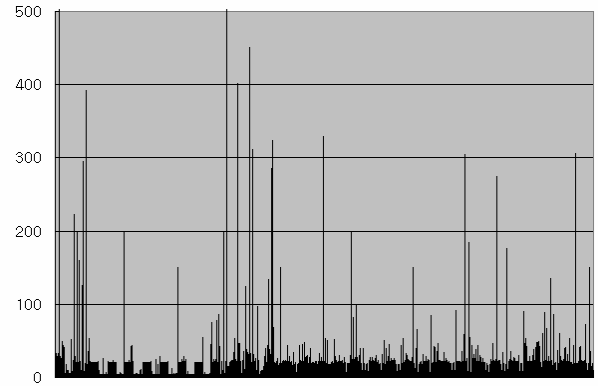


< 2.6.9- 인터럽트패치 +
자발적 선점 패치>

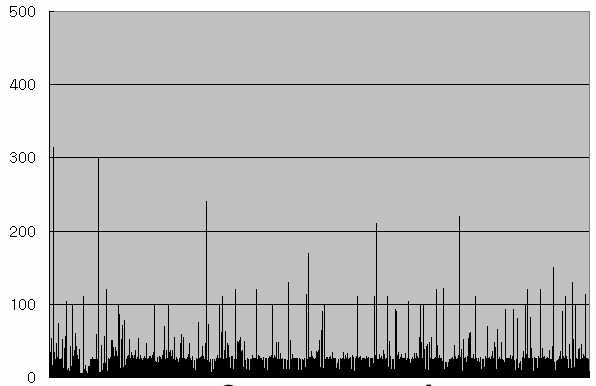
- interrupt disable time
  - cli~sti
- 인터럽트 지연시간을 가늠
- intlatency-2.6.patch적용
- perfview로 값을 저장함

성능 분석

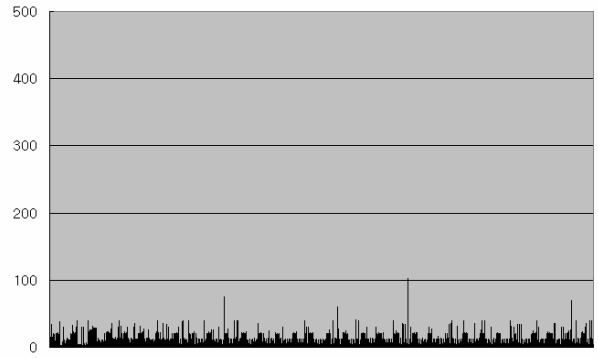
- 선점 지연시간 측정



< 2.6.9-vanilla >



< 2.6.9-인터럽트 패치 >



< 2.6.9- 인터럽트 패치 +
자발적선점 패치 >

결 론

- 다각적인 방법을 통한 리눅스에서의 실시간성 지원 노력
  - 이시간에도 다양한 패치들 개발됨
- 오픈 커뮤니티에서의 노력
- 100us 정도의 선점 지연시간을 보장하는 리눅스 커널
  - 리눅스의 장점을 살리면서도 실시간 성능이 크게 향상된 커널 사용 가능
  - Throughput과의 tradeoff
- 불안정한 측면 -> 꾸준한 안정화

Reference

- http://www.celinuxforum.org
- Understanding the Linux Kernel
- Linux Kernel Development
- Building Embedded Linux Systems
- Linux Kernel Internals
- http://redhat.com/~mingo/realtime-preempt/