효율적인 센서 운영체제를 위한 실시간 인터럽트 처리 기법

(A Real-time Interrupt Handling Scheme for Efficient Sensor Operating Systems)

안 재 훈 † 최 규 호 ††
(Jaehoon Ahn) (Kyuho Choi)
김 태 형 †† 홍 지 만 †††
(Taehyung Kim) (Jiman Hong)

요 약 무선 센서 네트워크가 적용되는 새로운 응용분야는 점점 더 정교하고 복잡한 태스크 수행과 정해진 시간 내에 태스크 수행을 완료해야 한다는 조건 만족을 요구하고 있다. 그러나 현재까지, 무선 센서 네트워크에서 센서 노드의 자원 제약성과 수행 작업 특성을 고려한 실시간 센서 운영체제 기반의 인터럽트 처리 기법에 관한 연구는 미비하다. 본 논문에서는 센서 운영체제에서 실시간성을 만족시키기 위한 요구 사항들을 분석하고, 이를 바탕으로 시스템을 설계 및 구현한다. 또한 다양한 검증을 통해, 제안하는 기법의 신뢰성 제공을 확인하며, 시뮬레이션을 통해 실시간 특성에 대한 요구사항 충족 및 성능의 효율성을 검증한다.

키워드 : 센서 운영체제, 인터럽트, 실시간 태스크

Abstract A new application area in which wireless sensor networks are applied requires the performance of more elaborated and complicated task and the completion of those tasks within a time limit. Until now, it is, however, insufficient to do research on the mechanism of handling interrupt based on real-time sensor operating systems which carefully consider the limitation of resources of sensor nodes and the property of tasks which is executed in a wireless sensor network area. In this paper, the requirements satisfying real-time in sensor operating systems are analyzed and based on this, a system is designed and implemented. In addition, the proposed mechanisms are confirmed by several verification methods, and the efficiency of the performance and the satisfaction of those requirements for real-time are verified by simulation.

Key words : Sensor Operating Systems, Interrupt, Real-time task

· 본 연구는 정보통신산업진흥원의 SW공학 요소기술 개발과 전문인력 양성 사업의 결과물임을 밝힘
· 이 논문은 2009 한국컴퓨터종합학술대회에서 '효율적인 센서 운영체제를 위한 실시간 인터럽트 처리 기법'의 제목으로 발표된 논문을 확장한 것임

† 정 회 원 : 숭실대학교 컴퓨터학과
corehun@gmail.com
†† 비 회 원 : 숭실대학교 컴퓨터학과
chlrbgh0@gmail.com
epsilon104@gmail.com
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
교신저자 jiman@ssu.ac.kr
논문접수 : 2009년 8월 13일
심사완료 : 2010년 1월 7일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제4호(2010.4)

1. 서 론

센서 네트워크의 연구 분야가 다양해지며, 정해진 시간 내에 작업을 완료하는 실시간성에 대한 요구도 높아지고 있다[1,2]. [3]에서는 환자 건강 상태 모니터링, 건축물의 붕괴 위험 감지와 같은 긴급 상황에 대처하기 위해 실시간 질의응답을 지원하는 통신 스케줄링 기법을 다룬다. 이와 같은 시스템은 다양한 형태의 메시지를 처리할 수 있어야 하며, 많은 양의 메시지가 동시 다발적으로 발생하기 때문에 이를 처리할 수 있는 효율적인 통신 프로토콜을 제공해야 한다. 본 논문에서는 기존 무선 센서 네트워크에서 실시간성 향상을 위한 노력들을 기반으로, 각 센서 노드들이 운영체제 수준에서 실시간 응용에 대한 지원이 용이하도록 실시간 지원 기법과 인터럽트 처리 구조의 설계 및 구현을 연구하였다.

센서 노드의 가장 큰 특징이면서 단점으로 자원 제약성을 들 수 있다. 대부분의 센서 노드는 비용 효율적 운영을 위해 저 사양 프로세서와 메모리가 탑재되며, 배터리를 통해 전원을 공급받기 때문에 에너지 효율성을 고려해야 한다. 특히 에너지 사용 부분은 센서 노드 수명과 직결되기 때문에 소형 풍력 발전기나 태양 전지를 사용하여 지속적으로 에너지를 공급하는 대안들이 제시되고 있다. 이러한 환경에서 에너지가 기준치 이하로 떨어졌을 때, 현재 상태를 저장하고, 추후 배터리가 재충전되면 이전의 상태를 복구하여 재동작하는 기법이 사용될 수 있다. 이 때, 무선 통신 또는 센싱 인터럽트로 노드의 현재 상태를 저장하거나 복구하는 작업이 방해받아서는 안된다[4]. 따라서 센서 운영체제 수준에서 실시간성 향상을 고려하여 긴급한 인터럽트가 먼저 처리될 수 있도록 센서 운영체제에서 실시간 인터럽트 구조와 실시간 기법의 지원이 필요하다.

본 논문의 구성은 다음과 같다. 본 연구의 필요성과 연구 범위를 제시한 1장의 서론에 이어, 2장에서는 기존 실시간 운영체제 기반 인터럽트 접근 특성에 대해 살펴본다. 3장, 4장에서는 본 논문에서 제안하는 센서 운영체제 기반 실시간 지원 기법과 인터럽트의 설계 및 구현에 대해 서술한다. 5장에서는 성능 평가를 통해 본 논문에서 제안한 시스템의 향상된 실시간성을 확인하고, 6장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

2.1 하드웨어적인 접근 방법

[5]에서는 센서 노드에 탑재되는 프로세서 구조를 멀티 스레드 환경에 적합하도록 설계하여, 시스템 처리량을 최대화하는 기법을 제안했다. 센서 응용에서 사용자에게 높은 응답성을 제공하고, 여러 작업들을 동시에 수행하도록 하기 위해서, 멀티 스레드 환경 제공에 대한 요구가 높아지고 있다. 그러나 센서 노드의 저 사양 프로세서는 단일 작업 처리에 최적화 되어 있기 때문에, 문맥 교환 시간 증가 및 스레드 간 의존성으로 인한 성능 저하 문제가 발생하고 있다. 이를 해결하기 위해서, 프로세서 구조는 스레드가 다음 실행에 필요한 명령어의 개수를 미리 스케줄러에게 알려주는 기법, 각 스레드마다 레지스터 윈도우를 별도로 할당하여 문맥교환 시간을 줄이는 기법, 동기화 레지스터를 사용하여 경쟁 상태를 줄이는 기법 등을 제공한다.

2.2 인터럽트 스케줄러

빈번하게 발생하는 인터럽트를 처리하는 동안 실제 필요한 작업들을 하지 못하는 인터럽트 오버로드(Interrupt Overload) 문제가 발생한다. 즉 하드웨어 인터럽트 처리는 태스크 수행보다 높은 우선순위를 가지고 있기 때문에 중요한 작업이 정해진 마감시간 내에 처리하지 못하는 문제가 발생할 수 있다. 특히 임베디드 시스템에서는 자원 제약적 특성으로 스레드 수준에서 인터럽트 오버로드를 줄이는 것이 어렵다. [6]에서는 이 문제를 해결하기 위해 하드웨어와 소프트웨어의 협업을 통한 인터럽트 스케줄러를 제안했다. 소프트웨어 인터럽트 스케줄러는 특정 인터럽트 처리가 끝나기 전에 다시 인터럽트가 발생하는 것을 막기 위해서, 한 번에 하나의 인터럽트가 처리될 수 있도록 설정한다. 예를 들어 타이머 인터럽트의 경우, 미리 설정된 간격에 따라 인터럽트가 발생하는 것이 아니라, 처리 후에 다음 발생 시간을 설정하는 방법을 사용한다(one-shot timer). 하드웨어 스케줄러는 프로세서와 인터럽트 컨트롤러 사이에서 각각의 인터럽트에 카운터 값을 설정하고, 카운터 값이 0이 된 경우에만 프로세서에게 인터럽트가 발생했다는 것을 알림으로써 빈번하게 인터럽트가 발생하는 것을 막아준다.

2.3 인터럽트 서비스 스레드

Windows CE에서는 우선순위에 따른 인터럽트 처리를 지원하기 위해서, 인터럽트 처리 루틴을 스레드에서 처리할 수 있는 구조를 운영체제 수준에서 제공하고 있다[7]. 인터럽트 서비스 스레드(Interrupt Service Threads)는 생성된 이후에 휴지(idle) 상태를 유지하다가 인터럽트 발생 시 커널에 의해 깨어나 해당 인터럽트에 대한 작업을 처리하고, 작업이 끝나면 이를 커널에게 알리고 다시 휴지 상태로 들어간다. Windows CE에서 인터럽트는 한 개 이상의 이벤트 객체와 연결되어 있는데, 커널은 인터럽트 발생 시, 이와 연결되어 있는 이벤트 객체를 활성화시킨다. 이벤트가 활성화되면 이를 기다리고 있는 인터럽트 스레드가 동작하고, 한 번의 동작이 끝나면 이벤트 객체를 다시 비활성화 시켜 줄 것을 커널에게 요청한다.

3. 시스템 요구 사항

3.1 무선 센서 네트워크 플랫폼

무선 센서 네트워크의 센서 노드는 저 성능 프로세서, 작은 크기의 메모리, 배터리를 통한 전원 공급과 같은 자원 제약성을 갖는다. 이러한 자원 제약 하에서 실시간 지원 시스템 구현을 위해서는 다음과 같은 추가적인 요구 사항을 만족시켜야 한다[8].

- 태스크 마감시간(deadline): 실시간 운영체제는 시간의 민감성과 태스크의 주기적 실행을 특성으로 갖는다. 하지만 실행 지연 등으로 불규칙한 수행이 빈번하게 일어난다.
- 정교한 타이머: 타이머 인터럽트 서비스 루틴은 시스템 전반에서 다양한 작업을 수행하며, 지연이 발생할 경우 응답성 및 대기시간 증가로 처리되지 못하는 인터럽트가 발생하게 된다.
- 실시간 인터럽트 처리: 센서 운영체제 인터럽트 서비스 루틴은 실시간성 및 신뢰성 제공에 많은 부분을 차지하며, 응답성 향상과 같은 실시간 특성 제공을 위해 실시간 인터럽트 서비스 루틴 실행을 보장해야 한다[9].

위와 같은 태스크 실행시간 및 마감시간 보장, 타이머 인터럽트의 정교함, 실시간 인터럽트 처리 지원 등이 추가적 요구사항을 만족함으로써 무선 센서 네트워크의 센서 운영체제 실시간성 향상에 많은 영향을 미치게 된다.

4. 실시간 지원 기법 설계 및 구현

본 절에서는 실시간 시스템 구현을 위한 요구사항과 플랫폼 특성을 바탕으로 실시간성 향상 기법 설계 및 구현 내용을 자세히 보인다.

4.1 실시간 보정

센서 운영체제는 일반적으로 타이머 인터럽트 서비스 루틴의 많은 양의 작업 처리로 시스템 전반에 지연과 주기 태스크의 규칙적 실행에 문제로 이어진다. 또한 ISR(Interrupt

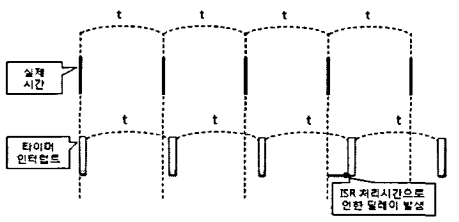

그림 1 ISR 지연

Service Routine)의 처리 지연은 실시간 작업 처리에 악영향을 준다. 그림 1은 실제 타이머 설정 주기와 ISR 실행 간격 간의 차이를 보인다. ISR 실행 지연은 순차적 인터럽트의 응답시간 증가로 이어진다. 센서 노드는 오랜 시간 재부팅 없이 동작하며 실시간 시스템의 경우 고 배율의 타이머를 사용하기 때문에, 작은 지연이 처리되지 못하는 인터럽트로 이어진다.

ISR 실행 지연과 실시간 보정을 위한 타이머 인터럽트 주기 설정이 필요하다. 고정적 주기 설정은 주기의 크고 작음에 따라 시스템 오버헤드 증가와 응답시간 증가로 이어진다. 따라서 ISR 실행 시간을 고려한 주기 설정이 필요하며 다음과 같이 설정한다.

$$T_{n+1} = T_n + (t - T_{ISR}) \quad : T_{ISR} = \text{ISR 수행시간}$$

이러한 주기 설정과 실시간 보정은 다음 타이머 인터럽트 주기를 커널 수준에서 직접 설정하며, 센서 운영체제의 타이머에 대한 시스템 반응 속도를 높임으로써 순차적 지연에 유연한 대처가 가능하다.

4.2 주기 태스크 구현

실시간 운영체제는 태스크의 주기적 동작 특성을 가지나, 모든 태스크의 주기적 특성 부여에는 어려움이 많다. 따라서 태스크 수행의 불규칙성과 응답시간 지연으로 시스템 반응 속도에 문제가 따르게 된다. 태스크의 규칙적 수행 및 응답시간 감소를 위해 주기(period)와 마감시간(deadline)을 스레드 생성 시 이용한다. 또한 주기 스레드 생성 및 주기적 동작을 위해 표 1의 각 정보를 TCB(Thread Control Block) 필드에 저장한다.

생성된 주기 태스크의 다음 주기 및 슬립 시간은 그림 2와 같이 설정되고, 현재 틱(current_tick)을 연산 기준으로, 정해진 시간에 깨어나 주기적인 동작이 가능하도록 설계한다.

표 1 주기 태스크 설정 정보

	설명
deadline	마감 시간
period	주기 시간
next_period	다음 주기까지 남은 시간
next_deadline	다음 종료까지 남은 시간
sleep_tick	슬립 시간

```
#ifdef PERIOD
void nos_set_period() {
   next_period   = current_tick + period;
   next_deadline = next_period + deadline;

   if ( (next_period < current_tick) && (next_period > period))
      next_period += period;
   if ( next_period > current_tick)
      sleep_tick = next_period - current_tick;
}
#endif
```

그림 2 주기 태스크 설정

4.3 O(1) 스케줄러

실시간 운영체제 스케줄러는 CPU 활용도, 처리율, 인터럽트 처리 시간 예측성, 태스크 마감시간 보장 등을 목표로 한다. 또한 최상위 우선순위 스레드를 찾기 위한 O(1) 복잡도 스케줄러가 필요하다[10,11].

본 논문에서는 8bit MCU 상에서 동작하는 센서 운영체제의 시스템 효율과 항상 일정한 동작시간을 고려한 O(1) 스케줄러 설계를 위해 32단계 우선순위 구분 및 스레드 당 하나의 우선순위를 갖도록 설계한다.

또한 제안하는 O(1) 스케줄러는 기존의 O(1) 스케줄러 특징 및 센서 운영체제의 특성에 따라 비교 수행 시 8bit 포인터 연산과 슬립 상태에서 깨어나는 스레드가 있을 경우에만 스케줄링을 진행한다. 이를 통해 시스템 반응속도와 응답시간 감소가 가능하게 된다. 최상위 우선순위를 찾기 위해 그림 3과 같이 _ready_priority_bit를 이용한 우선순위를 반환 과정을 거치도록 설계하였다.

4.4 실시간 인터럽트

인터럽트 처리에 대한 하드웨어적 지원이 부족한 환경에서는 효율적 수행을 보장하는 우선순위 기반 인터럽트 처리 기법이 필요하다. 그림 4와 같은 인터럽트 처리의 실시간성 향상을 고려한 순차 및 중첩 인터럽트 처리 구조를 제안하며, 사용자 모드 선택을 통해 유연성 및 효율성을 지원하도록 설계하였다.

4.4.1 우선순위 기반 순차 인터럽트 처리

순차 처리 구조는 발생된 인터럽트를 우선순위 기반

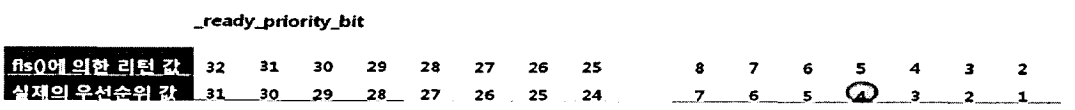

그림 3 O(1) 스케줄러 우선순위 변환

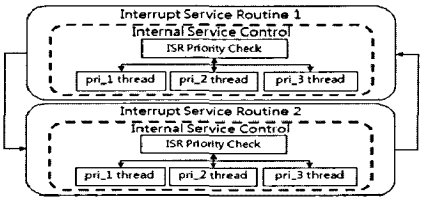

그림 4 실시간 인터럽트 처리 구조

```
void nos_seq_Interrupt0 {
  if(SELECT_MODE) { // 순차 및 중첩 ISR 모드
    if(Excute_BIT) { // 수행 인터럽트 여부 확인
      발생 인터럽트 seq_wait_queue 삽입
    } else { /* 발생 인터럽트 수행 */ }
  } else { /* 중첩 처리 루틴*/ }
}
```

그림 5 우선순위 기반 순차 인터럽트 처리 구조

정렬을 통해 최우선순위 인터럽트의 처리 지연 시간을 최소화한다. 제안한 순차처리 구조는 그림 5와 같으며, Execute_Bit 추가 및 우선순위 기반 seq_wait_queue 구성을 통해 인터럽트 대기시간 및 수행 제어를 한다.

4.4.2 우선순위 기반 중첩 인터럽트 처리

중첩 처리 구조는 현재 수행 중인 인터럽트의 우선순위에 기반 하여 선점 여부를 판별하고, 인터럽트 중첩 수행을 통해 실시간 인터럽트 처리를 지원한다. 중첩 인터럽트 처리를 위해 ISR_Pend(), ISR_Nest() 선점 처리 API와 ISR_Recover() 복구 처리 API 및 중첩 인터럽트 처리 과정에서 발생할 수 있는 태스크 블록킹(blocking), 태스크 대기(wait) 등을 고려한 뮤텍스(mutex) 및 우선순위 상속 기법 등을 제공한다. 우선순위 기반 중첩 인터럽트 동작은 그림 6과 같다.

```
void nos_nested_Interrupt0 {
  if(SELECT_MODE) { /* 순차 처리 루틴 */ }
  else { // 중첩 처리 루틴
    if( 발생 인터럽트의 우선순위가 높을 경우 ) {
      ISR Nesting User API (Pend, Nest, Recover)
    // 인터럽트 상태 저장, 중첩, 복귀 수행
    } else { /* 발생 인터럽트 seq_wait_queue 삽입 */ }
  }
}
```

그림 6 우선순위 기반 중첩 인터럽트 처리 구조

5. 성능 평가

본 절에서는 제안한 기법 검증 및 성능 평가를 수행하고, 시스템 실시간성 향상을 확인한다.

표 2는 본 논문에서 사용하는 실험 환경을 간략히 보이며, 그림 7은 태스크 증가에 따른 스케줄링 시간 비교 결과이다. 기존 커널은 태스크 증가에 따라 수행 시간의 선형적 증가를 보이지만, O(1) 스케줄러 및 주기 태스크 실시간 보정을 구현한 실시간 커널은 태스크 수와 관계없이 수행 시간 변동이 거의 없는 것을 확인할 수 있다.

표 2 실험 환경

플랫폼 구성	설명
운영체제	NanoOS-2.3.3
센서 보드	Ubi-Coin
CPU	MSP 430
RAM	4KB
ROM(FLASH)	128KB

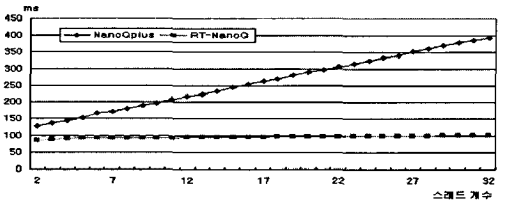

그림 7 스레드 개수 변화 시 스케줄링 시간

그림 8은 제안하는 커널 기반에서 동작하는 응용 프로그램과 기존 커널에서 동작하는 응용 프로그램의 응답 시간을 비교한 결과이다. 제안하는 커널에서 동작하는 응용은 기존 커널에 비해 응답 시간의 변화가 거의 없고 태스크의 주기적 특성을 보이고 있는 것을 확인할 수 있다.

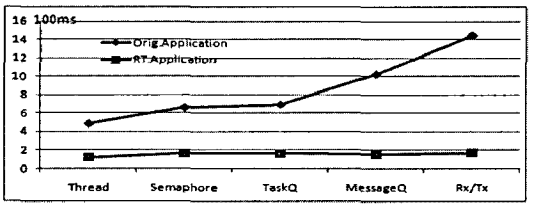

그림 8 응용 프로그램 응답 시간

실시간 인터럽트는 그림 9와 같이 사용자 설정 모드에 기반 하여 수행되며, (a)와 (b)는 순차 및 중첩 모드에 대한 수행 결과이다. 또한 실시간 인터럽트 성능 평가를 위해 그림 10과 같이 시뮬레이터를 구현하였으며, 실시간 보정, 주기 태스크, 순차 및 중첩 인터럽트의 시뮬레이션 환경을 제공한다.

```
ISR 3(pri=7) RUNNING:Q_stat[]
ISR 3(pri=7) RUNNING:Q_stat[2(10)]
ISR 3(pri=7) RUNNING:Q_stat[2(10),5(8)]
ISR 3(pri=7) RUNNING:Q_stat[9(12),2(10),5(8)]
ISR 9(pri=12) RUNNING:Q_stat[2(10),5(8)]
ISR 9(pri=12) RUNNING:Q_stat[2(10),5(8),6(4)]
ISR 2(pri=10) RUNNING:Q_stat[5(8),6(4)]
```

그림 9(a) 우선순위 기반 순차 인터럽트

```
ISR ID 1(pri=3) START
  ISR ID 6(pri=4) START
  ISR ID 6(pri=4) END
    ISR ID 5(pri=8) START
      ISR ID 2(pri=10) START
      ISR ID 2(pri=10) END
    ISR ID 5(pri=8) END
ISR ID 1(pri=3) END
```

그림 9(b) 우선순위 기반 중첩 인터럽트

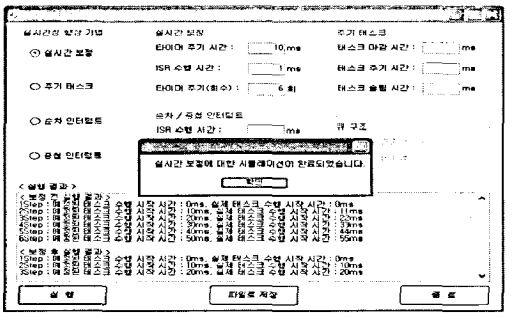

그림 10 AFITor 수행 화면

지금까지, 제안한 실시간 기법 검증을 통해 센서 운영체제의 실시간성 지원과 실시간 처리 기법 요구사항에 적합한 기능을 제공하는 것을 확인할 수 있었으며, 그림 11은 최상위 우선순위 기반 인터럽트 대기시간을 'AFITor' 시뮬레이터를 통해 측정한 결과이다. 본 논문에서 제안한 실시간 인터럽트 기법이 기존 인터럽트 처리 기법에 비해 인터럽트 처리 대기시간을 비교하였을 때 효율성이 높음을 확인할 수 있다.

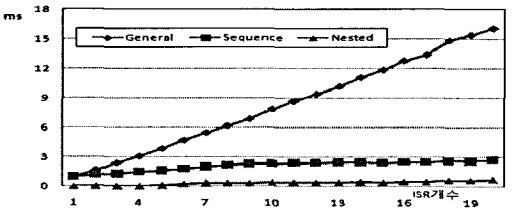

그림 11 ISR 개수 변화 시 ISR 대기 시간

제안한 시스템이 센서 노드용 실시간 처리 수행에 효율성을 해치지 않는 수준의 동작을 보장하는 것을 확인할 수 있다.

6. 성능 평가

본 논문에서는 실시간 센서 운영체제 설계에 필요한 시스템 프로토 타입의 설계 및 구현을 하였다. 또한 제안된 시스템을 실제 센서 운영체제에서 사용할 때 발생할 수 있는 제약 조건을 해결하고, 다각적인 방법을 통해 성능 검증을 위해 시뮬레이터를 사용하였다. 현재, 시스템 프로토 타입 및 시뮬레이터의 기능 개선 연구를 진행 중이며, 이 연구가 완료되면 다양한 응용 환경의 시뮬레이션 또한 가능할 것이다. 본 논문에서 제안한 시스템이 보다 안정적인 실시간 시스템으로 인정받기 위해 보완 및 해결해야 할 부분들이 남아 있으며, 향후 연구를 통해 보다 우수한 성능을 발휘할 수 있는 실시간 센서 운영체제를 구현할 수 있을 것이다.

참 고 문 헌

[1] E. Jafer, and K. Arshak, The development of wireless sensor system for pressure and temperature signals monitoring, the 1st international conference on Ambient media and systems, 2008.

[2] S. Mahlkecht, and S. Madani, On Architecture of Low Power Wireless Sensor Networks for Container Tracking and Monitoring Applications, 5th IEEE International Conference on Industrial Informatics, vol.2, pp.353-358, 2007.

[3] O. Chipara, C. Lu, and G. Roman, Real-Time Query Scheduling for Wireless Sensor Networks, 28th IEEE International Real-Time Systems Symposium, pp.389-399, 2007.

[4] Xinyu Feng, Zhong Shao, Yuan Dong, and Yu Guo. Certifying low-level programs with hardware interrupts and preemptive threads. In Proceedings of the 2008 ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI 2008), ACM Press, pp.170-182, Tucson, Arizona, Jun 2008.

[5] G. Hoover, F. Brewer, and T. Sherwood, A case study of multi-threading in the embedded space, the international conference on Compilers, architecture and synthesis for embedded systems, pp. 357-367, 2006.

[6] J. Regehr, and U. Duongsaa, Preventing interrupt overload, the ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, pp.50-58, 2005.

[7] Windows CE 6.0 - Interrupt service thread, http://msdn.microsoft.com/en-us/library/aa930165.aspx

[8] Gu, L. and Stankovic, J. A, t-kernel: providing reliable OS support to wireless sensor networks, the 4th international Conference on Embedded Networked Sensor Systems, pp.1-14, 2006.

[9] John Regehr, Nathan Cooprider, Interrupt Verification via Thread Verification, Electronic Notes in Theoretical Computer Science (ENTCS), vol.174 no.9, pp.139-150, June, 2007.

[10] Ingo Molnar, RT Patch, http://people.redhat.com/mingo/realtimeprempt/

[11] Ingo Molnar, http://lwn.net/Articles/102216/