신뢰성을 제공하는 센서 네트워크 운영체제 기법

김효승° 차호정
연세대학교 컴퓨터과학과
{hskim, hjcha}@cs.yonsei.ac.kr

Towards a Reliable Operating System for Wireless Sensor Networks

Hyoseung Kim° Hojung Cha
Dept. of Computer Science, Yonsei University

요 약

배포된 센서 네트워크에서 응용프로그램의 오류가 발생할 경우 시스템이 불안정하게 동작하거나 수행 불능 상태에 빠질 수 있다. 기존의 센서 네트워크 운영체제가 응용의 오류로부터 시스템을 보호하지 못함에 따라 본 논문에서는 신뢰성을 제공하는 센서 네트워크 운영체제 기법을 제안한다. 제안하는 기법은 소프트웨어 단계로 동작하여 특권모드와 페이징, 세그먼테이션 기능이 없는 하드웨어에서 사용 가능하며, 응용의 커널 및 다른 응용의 데이터 영역 훼손과 코드 실행, 하드웨어 직접 제어를 방지한다. 본 기법이 응용 오류로부터 시스템을 보호 가능함을 실제 구현과 평가를 통해 밝힌다.

1. 서론

센서 네트워크는 환경 모니터링이나 위치 추적과 같은 목적을 수행하기 위해 배포된 후 사용된다. 배치된 센서 노드에 응용프로그램의 오류가 발생할 경우 노드가 실행 불능 상태에 빠져 동적 코드 업데이트가 불가능해지거나, 오류가 발생했는지를 인식하지 못하고 잘못된 데이터를 수집할 수 있으므로 운영체제 차원의 신뢰성 제공 기법 개발이 요구된다.

기존의 대중적으로 사용되는 센서 네트워크 운영체제로는 TinyOS[1], SOS[2], Contiki[3]이 있다. TinyOS는 이벤트 기반의 운영체제로 커널과 응용이 정적으로 연결된 단일 코드 이미지를 생성한다. TinyOS에서 응용과 커널의 코드 및 데이터는 구분되지 않아 응용의 오류가 커널 수행에 직접적으로 영향을 미칠 수 있다. SOS와 Contiki는 역시 이벤트 기반의 운영체제이지만 개별 응용의 동적 업데이트를 위해 응용의 코드, 데이터 영역을 분리시켰다. 하지만 이들 운영체제도 응용이 커널이나 시스템 내 다른 응용의 데이터를 훼손하는 것을 막을 수 없다. 지금까지의 센서 운영체제들은 Watchdog timer를 통해 응용의 오류를 감지하고자 했으나 Watchdog timer는 오류 반복 발생, 응용 외부 메모리 훼손, 하드웨어 직접 제어 등의 상황에 대처하기 불충분하다.

센서 운영체제에서 신뢰성 제공을 위해서는 MMU(Memory Management Unit)와 특권모드가 없는 일반적인 센서 노드 디바이스를 위해 소프트웨어 단계의 안전 기법을 구동해야 한다. 기존의 소프트웨어를 이용한 시스템 보호 기법은 커널 확장이 가능한 범용 운영체제를 위해 연구되었다. 커널 확장 모듈은 범용 운영체제의 전통적인 보호 구간인 커널 주소 공간에서 실행되므로 확장 코드의 잠재적인 위험을 고려해야 했다. 대표적인 연구로는 SFI(Software Fault Isolation)[4]가 있다. 확장 모듈 에러로부터 시스템 안정성을 보장하기 위해 SFI는 확장 코드 중 간접 참조에 의한 데이터 기록 및 분기 코드에 대해 할당된 단일 세그먼트에서만 저장 또는 실행이 가능하도록 코드를 수정한다. 하지만 SFI 구동을 위해서는 세그먼테이션과 페이징이 가능한 하드웨어가 필요하다. 프로그래밍 언어 단계의 보호 기법으로는 C언어와 유사한 문법을 제공하는 Cyclone[5]이 있으나 C언어의 포인터나 배열 사용 문법을 변경해 적응에 어려움이 따르고 스택 안정성을 보장하기 위해서는 페이징이 필요하다.

본 논문에서는 기존 센서 네트워크 운영체제에서 제시된 바 없는 신뢰성을 제공하는 운영체제 기법을 제안한다. 커널의 시스템 안정성을 확보하기 위한 기능은 응용이 시스템 안정성을 해치는 시도를 감지하고, 시스템 내의 다른 응용의 수행에 영향을 미치지 않으면서 오류를 발생시킨 응용을 중단시킬 수 있다. 제안하는 운영체제 기법이 시스템에 신뢰성을 제공함을 상용 센서 디바이스에서의 실제 구현과 평가를 통해 밝힌다.

2. 신뢰성 제공을 위한 운영체제 기법

본 논문에서 제안하는 기법은 듀얼 모드 운용과 응용 코드 검사를 통해 응용 오류로부터 시스템 수행 불능 상태를 방지하고 응용들의 수행 독립성을 보장한다. 커널의 듀얼 모드 수행은 단일 선형 주소공간에서 커널과 응용의 메모리 공간을 논리적으로 분리한다. 특권 모드가 없는 하드웨어에서 듀얼 모드만 존재할 경우 응용의 커널 영역 침범을 막을 수 없으므로 응용은 빌드 단계에 정적 코드 검사 수행과 동적 검사 코드 삽입을 거쳐 신뢰성 있는 코드가 된다.

• 본 연구는 과학기술부에서 지원하는 국가지정연구실 사업으로 수행하였음 (과제번호 : 2005-01352).

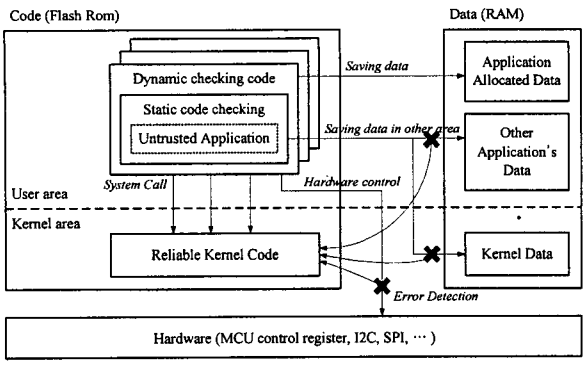

그림 1. 신뢰성을 제공하는 운영체제 기법의 개요

응용의 오류는 동적 검사 코드에 의해 커널에 보고되며, 커널은 오류가 발생한 위치를 사용자에게 알리고 해당 응용을 안전하게 종료시킨다. 그림 1은 신뢰성 제공을 위한 운영체제 기법의 개요를 나타낸다.

2.1 유저/커널 모드 수행

듀얼모드 수행의 목적은 단일 선형 주소공간에서 응용과 커널의 분리이다. 응용 코드 검사는 응용 자체 코드에 의한 문제만을 보호하므로 응용이 커널 코드 사용 도중 발생가능한 스택 오버플로우/언더플로우와 같은 문제를 방지할 수 없다. 따라서 시스템은 응용 코드검사로 잠재적 위험을 가지고 있는 응용으로부터 유저 공간을 보호하고, 커널 공간을 신뢰성 있는 커널 코드로만 접근한다.

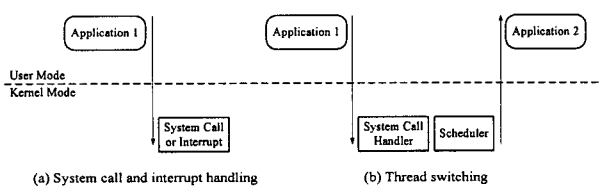

그림 2. 유저 모드와 커널 모드간의 전환

응용 수행은 대부분 유저 모드에서 이루어진다. 커널 모드로의 전환은 응용이 시스템콜을 호출하거나 시스템에 인터럽트가 발생했을 때 스택 전환으로 이루어진다. 그림 2.(a)는 응용이 커널에 의해 제공되는 서비스를 요청하기 위해 시스템콜을 실행할 때의 모드 전환을 나타낸다. 커널은 커널 모드로 전환한 뒤 응용의 요청 수행한다. 작업이 끝나면 시스템은 유저 모드로 복귀한다. 외부 시그널로부터 인터럽트 발생시 커널은 응용의 수행을 즉시 멈추고 커널 모드로 진입하여 인터럽트를 처리한다. 스케줄링은 커널 모드에서만 발생한다. 그림 2.(b)에서 보듯 블록을 요청하는 시스템콜 실행 시 커널은 스케줄러를 호출하여 다른 응용을 실행한다.

2.2 정적 코드검사 및 동적 검사코드 삽입

응용이 사용가능한 데이터와 코드 공간을 응용 내부로 한정시키고 응용 코드의 하드웨어 직접 제어를 막기 위해 응용 빌드 시 정적 코드검사와 동적 검사코드 삽입을 수행한다. 시스템이 안전하게 유지되기 위해서는 응용의 명령어가 자신에게 할당된 메모리 밖의 영역에 데이터를 저장해선 안되며, 응용 코드가 아닌 곳으로 분기해선 안되며, 시스템 제어 레지스터를 훼손해선 안된다. 이는 응용 명령어의 목적지(destination) 필드 검사로 해결 가능하다. 목적지 주소 검사를 통해 응용에 할당된 코드, 데이터 영역 밖의 접근과 memory-mapped 레지스터 접근을 막고, 목적지 항목이 무엇인지 검사해 MCU 상태 제어 레지스터 등의 접근을 막는다. 소스 필드 검사까지 수행할 경우 응용이 다른 응용의 메모리 공간이나 커널의 메모리 공간을 읽는 것 또한 막을 수 있으나, 이는 많은 오버헤드를 유발하고 시스템 안정성보다는 보안과 관련된 문제이다.

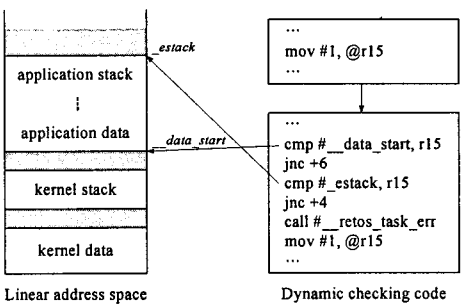

그림 3. 동적 검사 코드의 예 (TI MSP430)

정적 코드검사에서는 목적 필드에 직접 주소가 지정된 명령어 또는 PC-Relative jump와 같은 명령어에 대해 응용의 코드 또는 메모리 영역 밖으로 접근하는가를 평가하고 목적 필드가 하드웨어를 제어하는 레지스터인지 검사한다. 정적 검사에서 문제점이 발견될 경우, 해당 응용의 빌드는 실패하게 된다. 반면 명령어가 간접 주소 지정 방식(indirect addressing)일 경우, 해당 명령어 앞에 동적 검사 코드가 추가된다. 삽입된 코드는 명령어 수행 전에 목적 주소가 응용에 할당된 영역을 가리키는지 검사한 뒤 명령어를 실행한다. 영역 검사 시 할당된 영역이 아닐 경우 검사코드는 해당 응용의 에러를 커널에 보고한다. 커널은 해당 응용의 수행을 즉시 중지시킨다. 그림 3은 TI MSP430프로세서를 위한 동적 검사 코드의 예시다. 간접 주소 지정방식인 mov #1,@ r15는 r15 레지스터에 저장된 주소에 1을 기록하라는 명령이다. 삽입된 동적 코드는 r15 레지스터와 응용에 할당된 메모리 공간의 시작 및 끝 주소를 비교하여 적합 여부를 결정한다. r15가 응용 메모리 영역 밖을 가리킬 경우 동적 삽입코드는 명령어를 수행하지 않고 커널에 에러를 보고한다.

동적 검사코드의 비교 대상 영역은 명령어가 call, jump와 같은 코드 분기인지, store, move와 같은 데이터 저장인지에 따라 달라진다. 이를 통해 응용은 응용 코드 실행과 응용에 할당된 데이터 영역만을 수정 가능하며 응용의 커널 코드 실행 및 커널 데이터 변경이나 하드웨어 제어는 차단된다. 여기서 시스템콜은 예외이다. 범용 운영체제는 유저 인터럽트를 통해 시스템 콜을 구현하나 대부분의 마이크로컨트롤러에서는 이를 지원하지 않기 때문에 제안하는 운영체제 기법은 함수 콜을 이용해 시스템콜을 구현했다. 표 1은 응용 코드의 정적 및 동적 검사를 통한 커널 모드와 유저 모드에서의 접

근 권한을 보여준다.

	Kernel Mode	User Mode
Kernel Code	O	× (exc. system call)
Kernel Data	read/write	read only
Application Code	read/write	O (for current app.)
Application Data	read/write	O (for current app.)
Memory Mapped Register, Interrupt Vector	read/write	read only

표 1. 커널 모드와 유저 모드의 접근성

3. 평가

본 장에서는 제안하는 신뢰성 제공을 위한 운영체제 기법이 응용의 오류에 대해 기능적으로 작동함을 보인다. 평가는 8Mhz로 동작하며 10Kb의 RAM과 48Kb의 내부 Flash Rom을 갖춘 TI MSP430 f1611 마이크로컨트롤러를 장착한 Tmote Sky[6] 센서 노드 디바이스에서 진행되었다. 커널과 응용은 MS Windows 운영체제에서 Cygwin과 msp430-gcc를 이용해 구현하였다.

	Test Set
Stack safety	· General/Mutual recursive call

Data safety	· Directly addressed pointer
	int *tmp = 0x400; *tmp = 1;
	· Array indexing
	int array[10]; /* array in heap area */
	for(i = 10; i > 0; i--) array[i-100] = i;

Code safety	· Directly addressed function pointer
	void (*func)(void) = 0x1000; func();
	· Buffer overrun (damaging return address)
	void func() {
	int array[5], i;
	for(i = 0; i < 10; i++) array[i] = 0;
	}

Hardware safety	· Disable interrupt · Flash rom writing

표 2. 운영체제 신뢰성 테스트 방법

제안하는 운영체제 기법이 응용의 오류에 대처 가능함을 보이기 위해 응용의 오류에 대한 안정성 보장 구간을 Stack, Data, Code, Hardware로 나누었다. Stack safety는 응용의 함수 호출 또는 지역 변수 사용에 따른 스택 오버플로우 방지를 의미하며 Data safety는 시스템 내 다른 응용에 할당된 데이터 영역 훼손이나 커널 데이터 영역 훼손으로부터의 보호를 나타낸다. Code safety는 커널 코드나 시스템 내 다른 응용의 코드 실행을 막을 수 있는지 의미하며 Hardware safety는 응용의 하드웨어를 직접 제어로부터 안전한지를 나타낸다. 표 2는 각 안정성 보장 구간에 대한 테스트 방법이다. Stack safety는 일반 재귀호출과 상호 재귀호출을 통해, Data safety는 직접 주소 지정된 포인터와 범위가 벗어난 배열 인덱싱을 이용한 데이터 저장시의 응용 외부 메모리 훼손을 통해 테스트한다. Code safety는 직접 주소 지정된 함수포인터 호출시, 지역 변수 버퍼 오버플로우에 의한 함수 리턴 주소 훼손시 잘못된 코드를 실행하는지를 통해 점검한다. Hardware Safety에서는 응용이 시스템의 인터럽트를 불가능하게 만드는 코드와 Flash rom에 저장되어있는 다른 응용 또는 커널의 코드를 수정하는 코드를 이용했다.

테스트 결과 제안하는 운영체제는 표 2와 같은 시스템 안정성을 위협하는 응용 코드로부터 시스템 안정성을 보장할 수 있었다. 재귀호출, 범위가 벗어난 배열 인덱싱, 버퍼 오버플로우에 의한 함수 리턴 주소 훼손은 응용 실행시 동적 검사코드에 의해 커널이 이를 감지하고 실행을 중지시켰다. 직접 주소 지정된 포인터에 의한 데이터 저장이나 함수 호출, 인터럽트 불능이나 Flash rom 수정은 응용 컴파일 단계에 정적 코드 검사에 의해 검출되었다. 신뢰성을 보장하지 않는 기존 운영체제들 중 TinyOS에서 표 2의 에러를 포함한 응용을 구동한 결과 직접 주소 지정된 포인터 이용과 범위가 벗어난 배열 인덱싱 시 응용이 제대로 동작하지 않았으며 재귀호출, 직접 주소 지정된 함수포인터, 버퍼 오버플로우로 인한 리턴 주소 훼손, 인터럽트 불능, Flash rom 수정 실행시 시스템의 작동이 멈췄다. TinyOS는 시스템 중지 이후 watchdog timer가 동작하는 경우도 있었지만 상당수 시도에서 시스템 작동이 복구되지 않았다.

4. 결론

본 논문에서는 신뢰성을 제공하는 센서 네트워크 운영체제 기법을 제시하였다. 제안하는 기법은 듀얼모드 운영으로 커널과 응용의 수행 영역을 분리하고 응용 코드의 정적/동적 검사를 통해 응용에 할당된 영역 외부로의 접근을 제한한다. 제시하는 기법이 MMU와 특권모드가 없는 센서 디바이스에서 응용의 에러로부터 시스템을 보호할 수 있음을 실제 구현과 평가를 통해 보였다.

제안하는 기법을 이용한 운영체제는 아직 개발 단계에 있다. 본 논문을 통해 신뢰성 제공 기법이 센서 노드의 시스템 안정성을 보장함을 보였지만 커널의 잠재 버그 발견과 응용 작성을 위한 라이브러리 및 시스템콜 보완이 필요하다. 에너지 효율적인 네트워킹을 위한 네트워크 스택 성능 개선이 진행중에 있으며 센싱을 위한 디바이스 드라이버 개발, 타 프로세서로의 포팅 또한 이뤄지고 있다.

참고문헌

[1] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister . "System architecture directions for network sensors," *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Cambridge, MA, USA, November 2000

[2] Chih-Chieh Han, Ram Kumar Rengaswamy, Roy Shea, Eddie Kohler, Mani Srivastava., "SOS: A dynamic operating system for sensor networks," *Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys)*, Seattle, WA, USA, June 2005.

[3] Adam Dunkels, Björn Grönvall, Thiemo Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," *Proceedings of the First IEEE Workshop on Embedded Networked Sensors 2004 (IEEE EmNetS-I)*, Tampa, Florida, USA, November. 2004.

[4] Robert Wahbe, Steven Lucco, Thomas E. Anderson, Susan L. Graham, "Software-based fault isolation," *Proceedings of the Fourteenth ACM Symposium on Operating System Principles (SOSP)*, Asheville, NC, USA, December 1993.

[5] T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang. "Cyclone: A safe dialect of C," *Proceedings of the USENIX Annual Technical Conference*, Monterey, CA, June 2002.

[6] Tmote Sky, http://www.moteiv.com