(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2008년01월31일
(11) 등록번호 10-0799599
(24) 등록일자 2008년01월24일

(51) Int. Cl.
G06F 9/30 (2006.01) *G06F 9/06* (2006.01)
(21) 출원번호 10-2006-0039906
(22) 출원일자 2006년05월03일
심사청구일자 2006년05월03일
(65) 공개번호 10-2007-0107426
(43) 공개일자 2007년11월07일
(56) 선행기술조사문헌
KR1020050007906 A
(뒷면에 계속)

(73) 특허권자
연세대학교 산학협력단
서울 서대문구 신촌동 134 연세대학교
(72) 발명자
차호정
서울 용산구 이촌1동 삼성리버스위트아파트 103동 702호
김효승
서울 서대문구 창천동 100-26 화이트빌 205호
(74) 대리인
백남훈, 이학수

전체 청구항 수 : 총 5 항 심사관 : 이정호

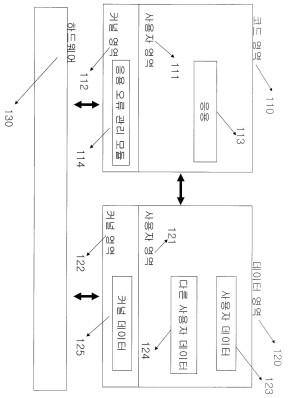
(54) 기억 관리 장치가 없는 마이크로 컨트롤러를 포함하는 임베디드 하드웨어의 운영 시스템 및 방법

(57) 요 약

본 발명은 커널과 응용이 논리적으로 분리되어 있고 정적 및 동적 코드 검사를 수행하는 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법에 관한 것이다.

이를 위해, 본 발명은 코드 영역, 데이터 영역 및 하드웨어로 포함하는 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템에 있어서, 상기 코드 영역과 데이터 영역은 커널의 듀얼 모드 수행에 의하여 사용자 영역과 커널 영역으로 나누어지고, 상기 코드 영역의 사용자 영역은 응용을 포함하고 상기 코드 영역의 커널 영역은 응용 오류 관리 모듈을 포함하여 구성되며, 상기 데이터 영역의 사용자 영역은 사용자 데이터와 다른 사용자 데이터를 포함하고 상기 데이터 영역의 커널 영역은 커널 데이터를 포함하여 구성되는 것을 특징으로 한다.

대표도 - 도1



(56) 선행기술조사문헌
KR1020050010640 A
US20060010314 A1
US6363436 B1
MMU가 없는 Thread기반 운영체제에서 스택 보호를 위한 메모리 관리 기법, 한국정보과학회 학술발표 논문집 제33권 제2호(A), 이영림 등, 2006. 10

특허청구의 범위

청구항 1

기억 관리 장치(MMU)가 없는 마이크로 컨트롤러를 포함하는 임베디드 하드웨어의 운영시스템에 있어서,

상기 임베디드 하드웨어의 코드 영역과 데이터 영역은 커널의 듀얼 모드 수행에 의하여 사용자 영역과 커널 영역으로 나누어지고,

상기 코드 영역의 사용자 영역은 응용 프로그램을 포함하고 상기 코드 영역의 커널 영역은 응용 오류 관리 모듈을 포함하여 구성되며,

상기 데이터 영역의 사용자 영역은 사용자 데이터를 포함하고 상기 데이터 영역의 커널 영역은 커널 데이터를 포함하여 구성되는 것을 특징으로 하는 임베디드 하드웨어의 운영 시스템.

청구항 2

청구항 1에 있어서, 상기 코드 영역은 플래시 롬으로 구성되는 것을 특징으로 하는 임베디드 하드웨어의 운영 시스템.

청구항 3

청구항 1에 있어서, 상기 데이터 영역은 램으로 구성되는 것을 특징으로 하는 임베디드 하드웨어의 운영 시스템.

청구항 4

응용 프로그램을 수행하던 스레드가 시스템 콜을 실행하거나 시스템의 인터럽트가 발생하는 (s201)단계;

상기 (s201)단계에서 시스템 콜이나 인터럽트가 발생하면 커널 모드로 전환하는 (s202)단계;

상기 응용 프로그램(113)을 가리키고 있던 스택 포인트(sp)를 커널 스택으로 옮기고 그 이후의 커널 수행을 계속하는 (s203)단계;

상기 커널에서의 수행을 모두 마치고 사용자 모드로 복귀하기 이전에 리턴 주소값 검사를 수행하여 올바른 리턴 주소값이라면 (s205)단계로 진행하고, 훼손된 리턴 주소값이라면 (s207)단계로 진행하는 (s204)단계;

상기 (s204)단계에서 검사된 주소값이 올바른 경우 사용자 모드로 복귀하는 (s205)단계와 상기 스레드의 수행을 계속하는 (s206)단계; 및

상기 (s204)단계에서 검사된 주소값이 훼손된 경우 응용 오류 관리 모듈(114)에 보고하는 (s207)단계와 상기 스레드의 수행을 종료하는 (s208)단계로 이루어지는 기억 관리 장치(MMU)가 없는 마이크로 컨트롤러를 포함하는 임베디드 하드웨어의 운영 방법.

청구항 5

청구항 4에 있어서, 상기 응용 프로그램은,

사용자가 작성한 응용 소스 코드를 어셈블리 코드로 컴파일하는 (s301)단계;

응용 프로그램 수행의 동적 검사를 위하여 동적 검사 코드를 삽입하는 (s302)단계;

상기 (s302)단계에서 동적 코드 삽입이 끝나면 바이너리 코드로 컴파일하는 (s303)단계;

상기 (s303)단계에서 컴파일된 바이너리 코드를 대상으로 정적 코드 검사를 수행하는 (s304)단계; 및

상기 (s304)단계에서 정적 코드 검사가 끝나면 응용 코드가 완성되는 (s305)단계로 이루어져 응용 프로그램이 빌드되는 것을 특징으로 하는 기억 관리 장치(MMU)가 없는 마이크로 컨트롤러를 포함하는 임베디드 하드웨어의 운영 방법.

명 세 서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

<15> 본 발명은 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법에 관한 것으로서, 더욱 상세하게는 커널과 응용이 논리적으로 분리되어 있고 정적 및 동적 코드 검사를 수행하는 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법에 관한 것이다.

<16> 일반적으로 마이크로 컨트롤러(Micro Controller)는 기억 관리 장치(Memory Management Unit, MMU)를 구비하여 복수의 프로그램에서 공통적으로 사용되는 기억 영역의 사용 상황을 관리하고 각 사용자 프로그램으로부터의 요구에 따라 기억 할당을 수행한다.

<17> 그러나 저비용, 저전력 등의 이유로 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드(embedded) 하드웨어의 사용이 증가하고 있으며, 특히 무선 센서 네트워크 분야에서 이러한 경향이 두드러진다.

<18> 그러나 마이크로 컨트롤러의 임베디드 하드웨어에서 기억 관리 장치가 존재하지 아니하므로 단일 선형 주소공간에서 특권 모드의 구동이 불가능한 환경의 발생하는 응용(application)의 오류는 커널(kernel)의 정상적인 수행을 방해한다. 여기서 응용은 응용 프로그램을 의미한다.

<19> 특히, 무선 센서 네트워크 분야와 같이 하드웨어를 외부 환경에 배치한 뒤 원격으로 관리하는 시스템에서는 더 심각한 문제가 있다. 응용의 오류 발생시 오류가 발생하였다는 사실을 인식하지 아니하고 계속하여 부정확한 데이터를 수집할 수 있으며, 노드가 수행 불능 상태에 이르게 되는 것이다.

<20> 이렇게 배포된 하드웨어가 수행 불능 상태에 빠지는 경우 이를 원격으로 복구하거나 새로운 응용을 탑재하는 것이 불가능하므로 전체 노드를 직접 수거하여 제거해야 하는 어려움이 있다.

<21> 결국 TI MSP430이나 AVR ATmega 등과 같은 마이크로 컨트롤러의 임베디드 하드웨어가 기억 관리 장치를 구비하지 아니하여 단일 선형 공간을 가짐으로써 응용의 오류가 발생하여 커널의 정상적인 수행을 방해하는 문제가 발생하는 것이다.

<22> 또한 상기 마이크로 컨트롤러의 임베디드 하드웨어에서 특권 모드가 존재하지 아니하여 응용이 자신에게 할당된 공간 외부 훼손이나 실행, 하드웨어 직접 제어하는 문제가 발생한다.

발명이 이루고자 하는 기술적 과제

<23> 따라서, 본 발명은 상기와 같은 문제점을 해결하기 위하여 발명한 것으로서, 본 발명은 커널이 듀얼 모드를 수행하여 단일 선형 주소 공간을 커널과 응용의 수행 공간으로 논리적으로 분리하는 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법을 제공하는데 목적이 있다.

<24> 또한 본 발명은 정적 코드 검사와 동적 코드 검사를 수행하여 신뢰성 있는 코드를 제공하는 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법을 제공하는데 목적이 있다.

<25> 또한 본 발명은 시스템에 적재된 각 응용이 자신의 주소 공간에 데이터를 기록하는 것은 허용되나, 응용 빌드 시 발견되지 않은 다른 응용 및 커널의 메모리 영역 침범이나 코드 실행, 하드웨어 직접 제어와 같은 응용 오류는 커널에 보고하는 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법을 제공하는데 목적이 있다.

발명의 구성 및 작용

<26> 이하, 첨부한 도면을 참조하여 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법을 상세히 설명하면 다음과 같다.

<27> 상기한 목적을 달성하기 위한 본 발명은 코드 영역, 데이터 영역 및 하드웨어로 포함하는 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템에 있어서, 상기 코드 영역과 데이터 영역은 커널의 듀얼 모드 수행에 의하여 사용자 영역과 커널 영역으로 나누어지고, 상기 코드 영역의 사용자 영역은 응용을 포함하고 상기 코드 영역의 커널 영역은 응용 오류 관리 모듈을 포함하여 구성되며, 상기 데이터 영역의 사용자 영역은 사용자 데이터와 다른 사용자 데이터를 포함하고 상기 데이터 영역의 커널 영역은 커널 데이터를 포함하

여 구성되는 것을 특징으로 한다.

<28> 더욱 바람직하게는 상기 코드 영역은 플래시 롬으로 구성되는 것을 특징으로 한다.

<29> 더욱 바람직하게는 상기 데이터 영역은 램으로 구성되는 것을 특징으로 한다.

<30> 상기한 목적을 달성하기 위한 본 발명은 응용 수행을 하던 스레드가 시스템 콜을 실행하거나 시스템의 인터럽트가 발생하는 (s201)단계; 상기 (s201)단계에서 시스템 콜이나 인터럽트가 발생하면 커널 모드로 전환하는 (s202)단계; 상기 응용(113)을 가리키고 있던 스택 포인트(sp)를 커널 스택으로 옮기고 그 이후의 커널 수행을 계속하는 (s203)단계; 상기 커널에서의 수행을 모두 마치고 사용자 모드로 복귀하기 이전에 리턴 주소값 검사를 수행하여 올바른 리턴 주소값이라면 s205)단계로 진행하고, 훼손된 리턴 주소값이라면 (s207)단계로 진행하는 (s204)단계; 상기 (s204)단계에서 검사된 주소값이 올바른 경우 사용자 모드로 복귀하는 s205)단계와 상기 스레드의 수행을 계속하는 (s206)단계; 및 상기 (s204)단계에서 검사된 주소값이 훼손된 경우 응용 오류 관리 모듈(114)에 보고하는 (s207)단계와 상기 스레드의 수행을 종료하는 (s208)단계로 이루어지는 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 방법이다.

<31> 더욱 바람직하게는 상기 응용은 사용자가 작성한 응용 소스 코드를 어셈블리 코드로 컴파일하는 (s301)단계; 응용 수행의 동적 검사를 위하여 동적 검사 코드를 삽입하는 (s302)단계; 상기 (s302)단계에서 동적 코드 삽입이 끝나면 바이너리 코드로 컴파일하는 (s303)단계; 상기 (s303)단계에서 컴파일된 바이너리 코드를 대상으로 정적 코드 검사를 수행하는 (s304)단계; 및 상기 (s304)단계에서 정적 코드 검사가 끝나면 응용 코드가 완성하는 (s305)단계로 이루어져 응용 빌드되는 것을 특징으로 한다.

<32> 이하, 첨부한 도면들을 참조하여 본 발명의 바람직한 실시예를 보다 상세하게 설명하고자 한다. 또한 도면상의 동일한 구성요소에 대해서는 동일한 참조부호를 사용하고 동일한 구성요소에 대해서 중복된 설명은 생략한다.

<33> 본 출원에서 사용한 용어는 단지 특정한 실시예를 설명하기 위해 사용된 것으로, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한 복수의 표현을 포함한다. 본 출원에서, "포함하다" 또는 "가지다" 등의 용어는 명세서상에 기재된 특징, 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

<34> 도 1은 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템의 구성도이고, 도 2는 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 방법의 순서도이다. 또한 도 3은 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법의 응용 빌드 순서도이고, 도 4는 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 방법에서의 응용 동적 코드 검사 및 정적 코드 검사가 필요한 명령어들을 나타낸다.

<35> 도 5는 TI MSP430 하드웨어에서의 동적 검사 코드의 실시예를 나타내고, 도 6은 TI MSP430 하드웨어에서의 정적 검사 코드의 실시예를 나타내고, 도 7과 도 8은 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법의 효용성을 평가하는 실험 결과를 나타낸다.

<36> 도 1을 참조하면, 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템은 코드 영역(110), 데이터 영역(120), 하드웨어(130)를 포함하여 구성된다. 여기서 상기 하드웨어(130)은 MCU(Micro-Controller Unit) control 레지스터, I2C, SPI 등을 포함할 수 있다.

<37> 상기 코드 영역(110)은 사용자 영역(111)과 커널 영역(112)로 나누어져 있으며, 사용자 영역(111)은 응용(113)을 포함하고, 커널 영역(112)은 응용 오류 관리 모듈(114)를 포함한다. 여기서 상기 코드 영역(110)은 플래시 롬(Flash Rom)으로 구성되는 것이 바람직하다.

<38> 또한 상기 데이터 영역(120)은 사용자 영역(121)과 커널 영역(122)로 나누어져 있으며, 사용자 영역(121)은 사용자 데이터(123)와 다른 사용자 데이터(124)를 포함하고, 커널 영역(122)은 커널 데이터(125)를 포함한다. 여기서 데이터 영역(120)은 램(RAM)으로 구성되는 것이 바람직하다.

<39> 이 경우 커널이 듀얼 모드를 수행하여 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템의 코드 영역(110)과 데이터 영역(120)을 사용자 영역(111,121)과 커널 영역(112,122)으로 논리적 분리시킨다.

<40> 상기 응용(113)은 정적 코드 검사와 동적 코드 검사를 위한 코드 삽입을 거친 상태로 상기 운영 시스템에 적재된다. 특히 상기 응용(113)이 자신의 사용자 데이터(123)에 저장하는 것은 올바른 동작이나, 다른 사용자 데이터(124)에 저장하거나 하드웨어(130)를 직접 제어하는 것은 응용 오류 관리 모듈(114)에 보고된다.

<41> 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템의 구성요소간의 연결관계는 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 방법을 통하여 보다 상세하게 살펴본다.

<42> 도 2를 참조하면, 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 방법은 (s201)단계에서 (s208)단계를 포함하여 구성된다.

<43> (s201)단계에서는 응용 수행을 하던 스레드(thread)가 시스템 콜(system call)을 실행하거나 시스템의 인터럽트(interrupt)가 발생한다. (s202)단계에서는 상기 (s201)단계에서 시스템 콜이나 인터럽트가 발생하면 커널 모드로 전환한다.

<44> (s203)단계에서는 응용(113)을 가리키고 있던 스택 포인트(stack point, sp)를 커널 스택으로 옮기고 그 이후의 커널 수행을 계속한다.

<45> (s204)단계에서는 커널에서의 수행을 모두 마치고 사용자 모드로 복귀하기 이전에 리턴 주소값 검사를 수행하여 올바른 리턴 주소값이라면 (s205)단계로 진행하고, 훼손된 리턴 주소값이라면 (s207)단계로 진행한다.

<46> 여기서 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템의 시스템 API는 일반적으로 단순히 함수 호출로 구현되기 때문에 사용자 스택에 커널 모드에서 복귀될 주소가 저장되며, 멀티 스레드 응용에서 다른 스레드가 상기 주소값을 훼손하여 문제를 발생시키므로 다른 스레드에 의하여 상기 응용(113)의 사용자 모드로의 복귀 주소값이 훼손되었는지 검사하는 것이다.

<47> (s205)단계에서는 (s204)단계에서 검사된 주소값이 올바른 경우 사용자 모드로 복귀하고, (s206)단계에서는 스레드의 수행을 계속한다.

<48> (s207)단계에서는 (s204)단계에서 검사된 주소값이 훼손된 경우 응용 오류 관리 모듈(114)에 보고하고, (s208)단계에서는 스레드의 수행을 종료한다.

<49> 도 3을 참조하면, 본 발명 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법의 응용 빌드 순서도를 나타내며, (s301)단계에서 (s305)단계를 포함하여 구성된다.

<50> (s301)단계에서는 사용자가 작성한 응용 소스 코드를 어셈블리 코드로 컴파일한다. (s302)단계에서는 응용 수행의 동적 검사를 위하여 동적 검사 코드를 삽입한다.

<51> (s303)단계에서는 상기 (s302)단계에서 동적 코드 삽입이 끝나면 바이너리 코드로 컴파일한다. (s304)단계에서는 상기 (s303)단계에서 컴파일된 바이너리 코드를 대상으로 정적 코드 검사를 수행한다.

<52> (s305)단계에서는 상기 (s304)단계에서 정적 코드 검사가 끝나면 응용 코드가 완성한다.

<53> 도 4를 참조하면, 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 방법에서의 응용 동적 코드 검사 및 정적 코드 검사가 필요한 명령어들을 나타낸다.

<54> 여기서 동적 코드 검사가 필요한 코드는 indirect addressing으로 목적 데이터를 변경하는 명령어(401)(예를 들어 mov 1, 1(r14)), 함수 리턴 명령어(402)(예를 들어ret, reti), 스택 푸쉬(push) 명령어(403)(예를 들어 push)이다. 이러한 명령어들(401,402,403)에 대한 동적 검사 코드를 원래 응용(113)의 어셈블리 언어에 추가하여 빌드한다.

<55> 또한 정적 코드 검사가 필요한 코드는 인터럽트 상태 제어 명령어(404)(예를 들어 eint, dint), PC-Relative 점프 명령어(405)(예를 들어 jmp +38), 목적 필드가 MCU 상태 제어 레지스터인 명령어(406)(예를 들어 mov 0, r2 /* r2는 msp430의 status reg. */)이다. 여기서 응용 코드가 컴파일된 후 완성된 바이너리 코드를 대상으로 이러한 명령어들(404,405,406)에 대한 정적 코드 검사를 수행한다.

<56> 도 5를 참조하면, TI MSP430 마이크로 컨트롤러가 장착된 센서 노드를 위한 동적 검사 코드 중 indirect addressing을 사용하는 call 명령어에 대한 실시예를 나타낸다.

<57> (501)에서 C언어에서 함수 포인터를 이용한 함수 호출 명령을 작성할 경우 어셈블리 언어로 컴파일하면 (502)처

럼 indirect addressing을 이용한 call 명령으로 나타난다. 여기서 indirect addressing call은 동적 검사가 필요한 코드이므로 (503)처럼 call 명령어 앞에 목적 필드를 검사하는 명령어가 추가되며, 상기 명령어들은 (504)처럼 응용에 할당된 코드 영역인지 범위 검사를 통해 확인할 수 있다.

<58> 도 6을 참조하면, MSP430 시스템에서 정적 코드 검사 중 call과 mov 명령어를 검사하는 실시예를 나타낸다.

<59> 바이너리 코드(601)에서 우선 call 명령어가 주소값(예를 들어 0x1254)을 호출하고 있다. 우선 (603)에서 상기 주소값이 Relocation Table(602)에 존재하는지 검사한다. 상기 Relocation table(602)에는 대부분의 컴파일러에서 컴파일 시 자동으로 생성 가능한 것으로 응용 코드가 사용하는 모든 심볼 정보와 해당 심볼을 참조하는 명령어의 오프셋이 나타나 있으므로 목적 주소가 Relocation Table(602)에 없을 경우 올바른 코드가 아님을 알 수 있다.

<60> 그 후 (604)에서는 해당 주소가 코드 영역 내에 존재하는지 범위 검사를 수행하고 이러한 (603)과 (604) 과정을 통과하면 call 명령의 검사가 종료된다. 또한 mov 명령어도 call 명령어처럼 (603)과 (604) 과정을 거치면 되나, (604)에서 코드 영역 대신 데이터 영역을 검사한다.

<61> 도 7과 도 8은 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법의 효용성을 평가하는 실험 결과를 나타낸다.

<62> 도 7을 참조하면, TI MSP430 마이크로 컨트롤러의 임베디드 하드웨어에서 오류 코드를 수행한 결과 스택, 데이터, 코드, 하드웨어 안정성에 있어서 만족함을 보여준다.

<63> 구체적으로 살펴보면, 도 7에서 기억 관리 장치가 없는 시스템에서 발생할 수 있는 응용의 오류들에 대한 안정성 보장 구간을 Stack, Data, Code, Hardware의 4단계로 나누었다. 여기서 본 발명에 따른 방법은 기계어 레벨의 검사를 수행하므로 응용프로그램을 작성하는 프로그래밍 언어와 무관하나, 여기서는 C언어를 기반으로 한 오류 코드를 이용하였다.

<64> Stack safety는 응용의 함수 호출 또는 지역변수 사용에 따른 스택 오버플로우 방지를 의미하며, 일반 재귀호출과 상호 재귀호출로 검사한다. Data safety는 시스템 내 다른 응용에 할당된 데이터 영역 훼손이나 커널 데이터 영역 훼손으로부터의 보호를 나타내며, 직접 주소 지정된 포인터와 범위가 벗어난 배열 인덱싱을 이용한 응용 외부 영역을 훼손하는 코드로 검사한다.

<65> Code safety는 커널 코드나 시스템 내 다른 응용의 코드 실행을 막을 수 있는지 의미하며, 직접 주소 지정된 함수 호출과 버퍼 오버플로우로 인한 함수의 리턴 주소값을 손상시키는 코드로 검사한다. Hardware safety는 응용의 하드웨어를 직접 제어로부터 안전한지를 나타내며, 응용이 시스템의 인터럽트를 불가능하게 만드는 코드와 플래쉬 롬에 저장되어있는 다른 응용 또는 커널의 코드를 수정하는 코드로 검사한다.

<66> 이러한 테스트 결과, 본 발명에 따른 시스템에서는 도 7의 시스템 안정성을 위협하는 응용 코드로부터 시스템 안정성을 보장할 수 있었다. 여기서 재귀호출, 범위가 벗어난 배열 인덱싱, 버퍼 오버플로우에 의한 함수 리턴 주소 훼손은 응용 실행 시 동적 검사코드에 의해 커널이 이를 감지하고 실행을 중지시켰다. 또한 직접 주소 지정된 포인터에 의한 데이터 영역 훼손이나 함수 호출, 인터럽트 불능이나 플래쉬 롬 수정은 응용 컴파일 단계에 정적 코드 검사에 의해 검출되었다.

<67> 도 8을 참조하면, 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 방법을 무선 센서 네트워크에 적용했을 때, 일반적인 센서 응용을 대상으로 한 성능 오버헤드 정도(801)와 응용 코드 크기의 오버헤드 정도(802)를 나타낸다.

<68> (801)에서 동적 검사 코드로 인한 수행 오버헤드는 0.5~8.4%로 나타났다. 구체적으로는 계산량이 많은 응용인 MPT_mobile과 같은 응용에서는 2%의 오버헤드를, 계산시간보다는 데이터 영역 접근이 많은 R_recv, Pingpong, Surge 같은 응용에서는 6~8%의 오버헤드를 기록하였다. 이를 통해 메모리 사용량이 많은 응용에서 보다 많은 성능 오버헤드가 발생할 수 있지만 그 정도가 크지 않음을 알 수 있다

<69> (802)에서 동적 검사 코드의 추가로 인한 코드 크기 증가량은 4~12%로 나타났다. 그러나 이 정도의 코드 크기 증가는 센서네트워크에서 널리 사용되는 TI MSP430이나 AVR ATmega128L과 같은 마이크로 컨트롤러의 플래쉬 롬에서 무리가 되지 않는다.

<70> 이러한 오버헤드의 정도를 볼 때, 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 이 기억 관리 장치가 없는 마이크로 컨트롤러 시스템에서 매우 적은 오버헤드로 동작한다는

것을 알 수 있다.

<71> 본 발명은 다양한 변경을 가할 수 있고 여러 가지 형태를 가질 수 있는 바, 특정 실시예들을 도면에 예시하고 본문에 상세하게 설명하고자 한다. 그러나, 이는 본 발명을 특정한 개시 형태에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물 내지 대체물을 포함하는 것으로 이해되어야 한다.

발명의 효과

<72> 이상에서 설명한 바와 같이, 본 발명에서 제안하고 있는 것은 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법으로서, 이러한 본 발명에 의하면 다음과 같은 효과를 얻을 수 있다.

<73> 첫째, 본 발명은 커널이 듀얼 모드를 수행하여 단일 선형 주소 공간을 커널과 응용의 수행 공간으로 논리적으로 분리함으로써 응용의 오류를 방지하여 커널이 정상적인 수행을 할 수 있도록 한다.

<74> 둘째, 본 발명은 정적 코드 검사와 동적 코드 검사를 수행하여 신뢰성 있는 코드를 제공함으로써 특권 모드가 존재하지 아니하여 응용이 자신에게 할당된 공간 외부 훼손이나 실행, 하드웨어 직접 제어를 방지할 수 있다.

<75> 셋째, 본 발명은 시스템에 적재된 각 응용이 자신의 주소 공간에 데이터를 기록하는 것은 허용되나, 응용 빌드 시 발견되지 않은 다른 응용 및 커널의 메모리 영역 침범이나 코드 실행, 하드웨어 직접 제어와 같은 응용 오류는 커널에 보고함으로써 커널은 오류가 발생한 위치를 알 수 있어 안전하게 응용을 종료할 수 있다.

<76> 결국 본 발명은 AVR ATmega, TI MSP430 등의 기억 관리 장치가 없는 저비용 마이크로 컨트롤러 급의 임베디드 기기의 안정성 보장하고, 무선 센서 네트워크에 적용 시 노드의 데이터 수집에서 시스템 안정적인 수행으로 데이터의 신뢰도를 높이고, 배포된 센서 노드의 응용 오류 시 시스템 자체적으로 오류 탐지 및 복구하는 효과가 있다.

도면의 간단한 설명

<1> 도 1은 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템의 구성도이다.

<2> 도 2는 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 방법의 순서도이다.

<3> 도 3은 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법의 응용 빌드 순서도이다.

<4> 도 4는 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 방법에서의 응용 동적 코드 검사 및 정적 코드 검사가 필요한 명령어들이다.

<5> 도 5는 TI MSP430 하드웨어에서의 동적 검사 코드의 실시예이다.

<6> 도 6은 TI MSP430 하드웨어에서의 정적 검사 코드의 실시예이다.

<7> 도 7과 도 8은 본 발명에 따른 기억 관리 장치가 없는 마이크로 컨트롤러의 임베디드 하드웨어에서 운영 시스템 및 방법의 효용성을 평가하는 실험 결과이다.

<8> <도면의 주요 부분에 대한 부호의 설명>

<9> 110 : 코드 영역 120 : 데이터 영역

<10> 111 : 사용자 영역 121: 사용자 영역

<11> 112 : 커널 영역 122 : 커널 영역

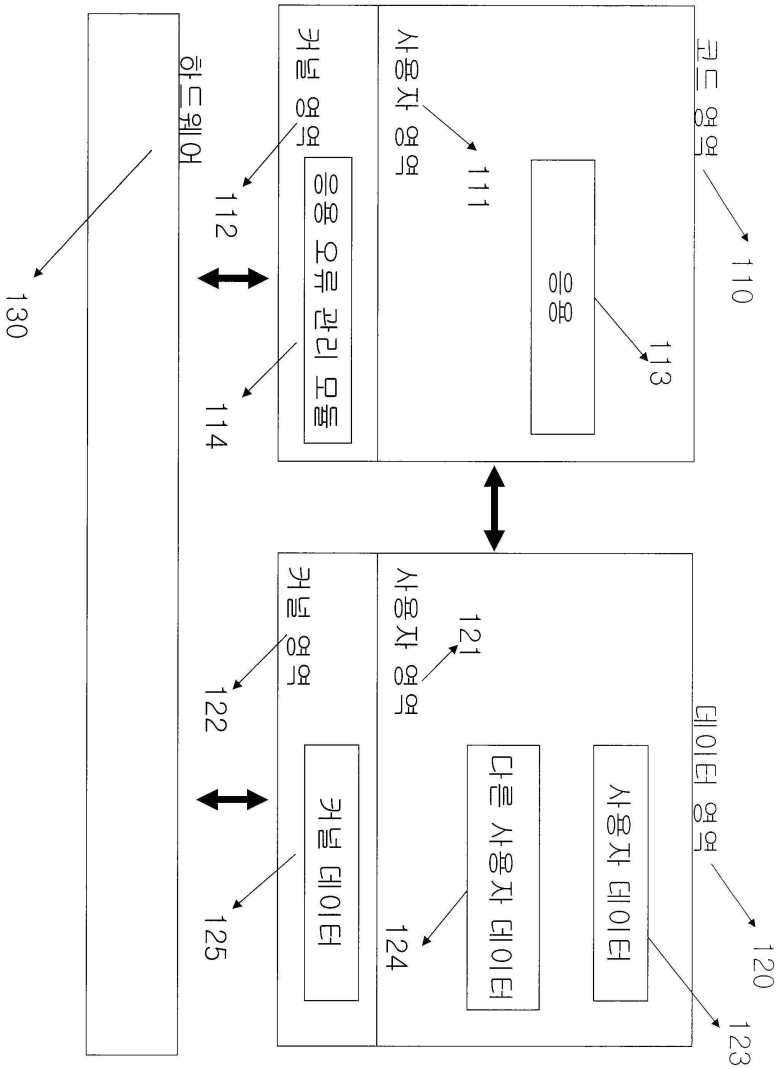
<12> 113 : 응용 123 : 사용자 데이터

<13> 114 : 응용 오류 관리 모듈 124 : 다른 사용자 데이터

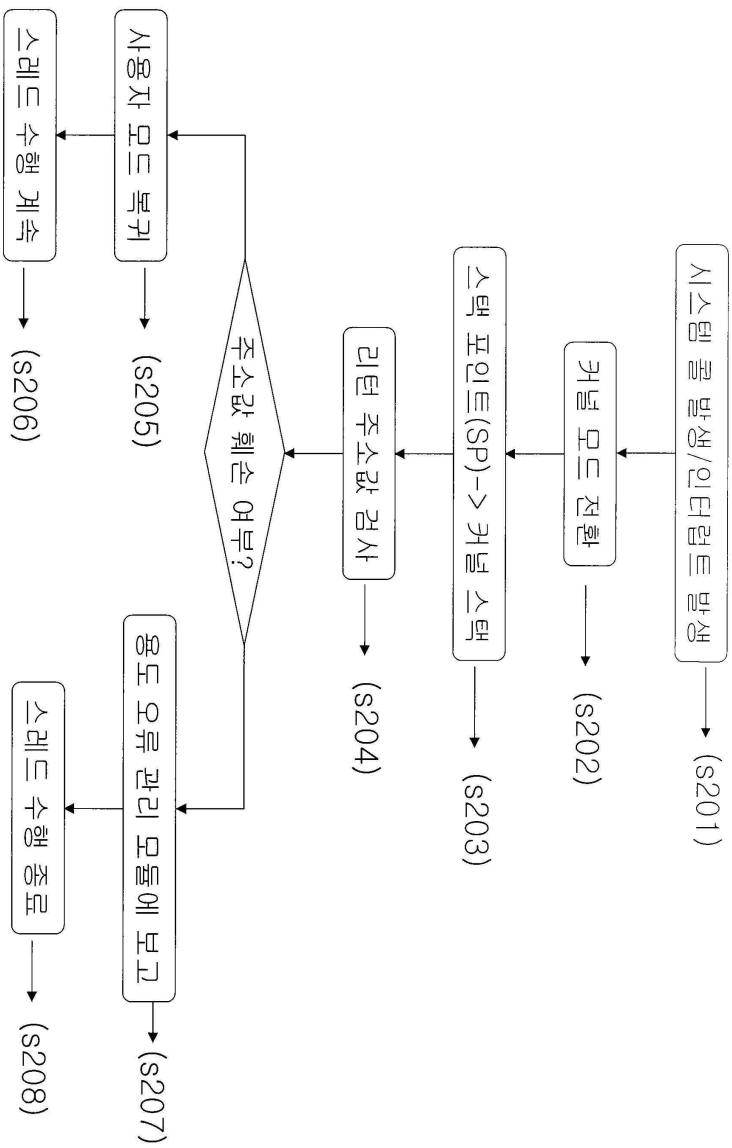
<14> 125 : 커널 데이터 150 : 하드웨어

도면

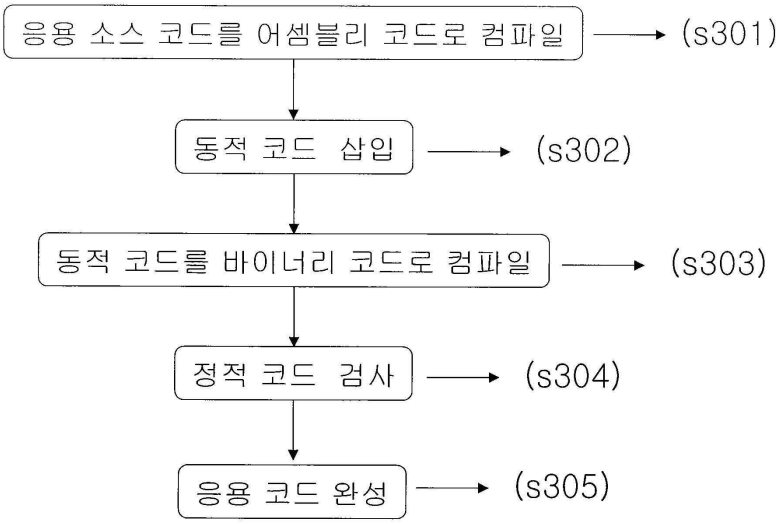
도면1


코드 영역
110
사용자 영역
111
응용
113
커널 영역
112
응용 오류 관리 모듈
114
하드웨어
130
데이터 영역
120
사용자 영역
121
사용자 데이터
123
다른 사용자 데이터
124
커널 영역
122
커널 데이터
125

도면2


시스템 콜 발생/인터럽트 발생
(s201)
커널 모드 전환
(s202)
스택 포인트(SP)-> 커널 스택
(s203)
리턴 주소값 검사
(s204)
주소값 훼손 여부?
사용자 모드 복귀
(s205)
스레드 수행 계속
(s206)
응용 오류 관리 모듈에 보고
(s207)
스레드 수행 종료
(s208)

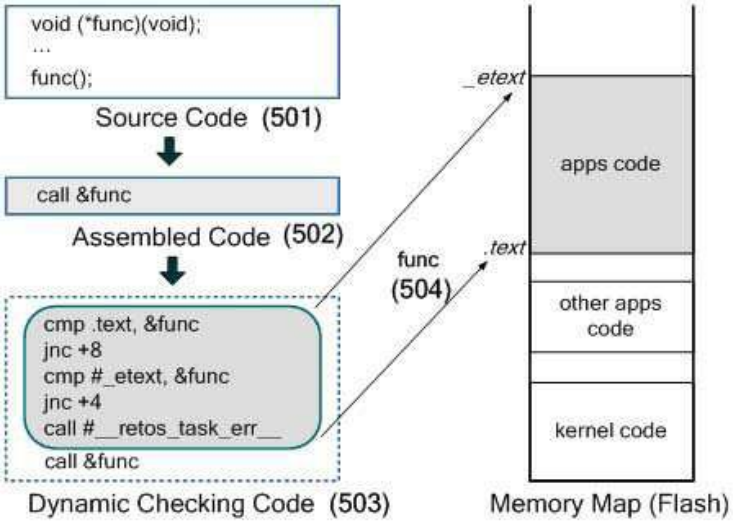
도면3



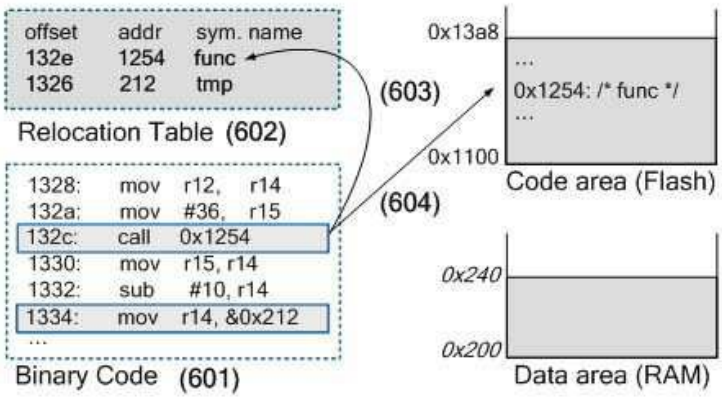
도면4

동적 코드 검사	정적 코드 검사
indirect addressing 으로 목적 데이터 변경 (401) 예) mov 1, 1(r14)	인터럽트 상태 제어 (404) 예) eint, dint
함수 리턴 명령어 (402) 예) ret, reti	PC-relative 점프 (405) 예) jmp + 38
스택 push 명령어 (403) 예) push	목적필드가 MCU 상태제어 레지스터 (406) 예) mov 0, r2 /* r2 는 msp430 의 status reg. */

도면5


void (*func)(void);
...
func();
Source Code (501)
call &func
Assembled Code (502)
cmp .text, &func
jnc +8
cmp #_etext, &func
jnc +4
call #__retos_task_err__
call &func
Dynamic Checking Code (503)
func
(504)
_etext
.text
apps code
other apps code
kernel code
Memory Map (Flash)

도면6


offset addr sym. name
132e 1254 func
1326 212 tmp
Relocation Table (602)
1328: mov r12, r14
132a: mov #36, r15
132c: call 0x1254
1330: mov r15, r14
1332: sub #10, r14
1334: mov r14, &0x212
...
Binary Code (601)
(603)
(604)
0x13a8
...
0x1254: /* func */
...
0x1100
Code area (Flash)
0x240
0x200
Data area (RAM)

도면7

	Test Set
Stack safety	- General/Mutual recursive call void foo() { foo(); }

Data safety	- Directly addressed pointer
	int *tmp = 0x400; *tmp = 1;
	- Array indexing
	int array[10]; /* array in heap area */
	for(i = 10; 1 > 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 asm volatile ("dint"); - Flash rom writing (memory mapped regs.) FCTL1 = FWKEY+ WRT;

도면8

Performance Overhead (801)	No check	Dynamic check	Overhead(%)
MPT_backbone	40326	40508	0.5
MPT_mobile	386566	394624	2.1
R_send	24815	26176	5.5
R_recv	4704	5010	6.5
Sensing	1056	1096	3.8
Pingpong	1243	1347	8.4
Surge	58723	62688	6.7

Code Size Overhead (802)	No check (byte)	Dynamic check (byte)	Overhead(%)
MPT_backbone	614	682	11.1
MPT_mobile	8726	10046	15.1
R_send	946	1014	7.2
R_recv	902	1004	11.3
Sensing	774	835	7.9
Pingpong	478	510	6.7
Surge	1436	1610	12.1