적응형 센서네트워크를 위한 커널 모듈화 기법

신효정° 차호정
연세대학교 컴퓨터과학과
{hjshin, hjcha}@cs.yonsei.ac.kr

Kernel Modulation Technique for Adaptive Wireless Sensor Networks

Hyojeong Shin° Hojung Cha
Dept. of Computer Science, Yonsei University

요 약

센서 네트워크 운영체제는 다양한 실험과 작업을 위해 다양한 기능을 수행해야한다. 이를 위해 센서 운영체제는 필요에 따라 응용프로그램을 설치하여 사용하고 이를 필요에 따라 수정하는 것을 지원한다. 이에 따라 센서 운영체제는 다양한 응용프로그램을 사용할 수 있도록 범용화 되고 고정된다. 또한 운영체제와 응용프로그램의 분리에 따라 오버헤드가 발생하게 된다. 따라서 센서 운영체제가 설치 된 운영체제에 따라 커널의 구조를 최적화 하고 응용프로그램 수행에 따른 오버헤드를 최적화 하는 작업이 필요하다. 본 논문에서는 커널을 모듈화 하고 필요에 따라 선택적으로 커널을 재구성할 수 있는 프레임워크를 제안한다. 제안하는 시스템은 모듈화 된 커널 자원을 효율적으로 관리하며, 수행하는 응용 프로그램의 요구 기능에 따라 커널의 구성을 바꿀 수 있어 최적화된 커널을 유지할 수 있다.

1. 서론

Wireless Sensor Networks에는 환경 모니터링, 이동 물체 추적 등 다양한 응용프로그램이 존재한다. 또한 이들 응용 프로그램이 수행하기 위해 가정하는 localization, tim sync, data aggregation, radio communication 등 다양한 센서 네트워크 기반 기술과 기타 하드웨어 드라이버 등이 존재한다. 이를 수행하기 위해 센서 운영체제는 그동안 응용프로그램과 하드웨어 드라이버를 하나의 이미지로 실행하는 방법[1]에서 운영체제와 응용프로그램을 분리하여 기능별로 응용프로그램을 작성, 재배포하는 방법[2][3]으로 발전해 왔다. TinyOS[1]가 모듈화 된 프로그래밍을 지원하지만 모듈 단위로 업데이트할 수없는 점을 보완하기 위해 Maté [3]는 센서 OS에 VM 을 설치하고 VM 에서 업데이트 가능한 모듈을 설치하여 실행하는 방법을 사용한다. Magnet OS[5] 는 센서 OS에 Java VM 을 탑재하였다. SOS[2]는 Module 프로그래밍을 지원하는 센서 네트워크용 운영체제의 좋은 예이다. SOS 는 MMUless system 의 한계를 극복하기 위해 모든 코드 내 사용하는 변수 및 함수의 메모리 주소를 text area 또는 stack base로부터의 상대 주소로 표현하고 사용 시 상대 주소를 통해 사용한다. Contiki [6] 는 프로그램 단위의 재 프로그램을 지원하는 센서 네트워크 용 스레드 베이스 OS이다. VM★[7]는 VM 용으로 컴파일 된 바이너리와 함께 이 바이너리에 최적화 된 VM를 함께 배포하는 미들웨어 수준의 방법을 지원한다. 이 같은 다양한 방법을 통해 보다 센서네트워크 운영체제는 다양한 응용 프로그램을 수행할 수 있게 되었고, 응용 프로그램의 수정 및 재배포가 용이하다는 점 등의 이점을 얻었다. 하지만 운영체제와 응용프로그램을 분리함에 따라 운영체제와 응용프로그램 사이의 소통상의 오버헤드가 발생하게 되었으며 운영체제 자체 운용비용이 오버헤드로 남게 되었다. 또한 센서 운영체제는 범용 운영체제의 성격을 가지게 되면서 다양한 하드웨어 드라이버와 다양하고 새로운 기능을 추가할 수 있어야 했는데, 기존의 센서 운영체제는 운영체제의 커널이 고정되어 있어 새로운 기능을 추가하기 위해 커널 전체를 수정하여 재배포하여야 하는 등의 한계를 가지고 있다. 이러한 환경에서 응용프로그램의 버전업그레이드나 최적화 역시 응용프로그램 안에서만 가능하게 된다.

커널의 기능을 재구성 하는 방법은 임베디드 시스템이나 전통적인 시스템에서 기원한다. LINUX 의 임베디드 버전인 uClinux 는 MMU가 없는 마이크로 컨트롤러 에 LINUX kernel 을 포팅 한 것으로 LINUX 가 가지는 여러 가지 기능은 물론 loadable module 을 지원한다. uClinux 는 MMU가 없는 환경에서의 할당 받은 메모리의 위치에 따라 code 의 주소가 달라지는 문제를 relocation과 Position Independent Code(PIC)를 이용하여 효과적으로 극복한다. uClinux 는 임베디드 환경에서 운영체제의 기능을 확장하는 관점에서 매우 성공적인 구현이지만, 센서 노드의 성능상의 제약으로 uClinux를 센서 노드에 그대로 활용할 수 없다.

본 논문에서는 응용 프로그램과 커널을 분리하여 운용할 뿐 아니라 커널을 모듈화 하고 필요에 따라 커널을 재구성할 수 있는 시스템을 제안한다. 제안된 시스템은 범용 운영체제로써 다양한 응용프로그램을 수행할 수 있을 뿐 아니라 응용프로그램 별로 최적화 된 커널 모듈을 탑재함으로써 운영체제 운용 따른 오버헤드를 최소화할 수 있다. 또한 응용프로그램이 필요로 하는 커널 기능만 보유하고 있을 수 있어 자원이 제약되어 있는 센서 네트워크 환경에서 운영체제의 자원 사용을 줄이고 응용프로그램에게 보다 많은 자원을 남겨 줄 수 있는 장점이 있다. 더불어 새로운 하드웨어 도입이나 커널 버그 및 기능 수정 시 응용 프로그램의 변경이나 지원 없이 변경 된 커널만 추가로 재배포할 수 있다. 이 시스템은 센서 네트워크 범용 운영체제인 RETOS[4] 위에 구현되었다. RETOS 는 모듈화 된 프로그램 및 멀티스레딩를 지원하는 센서 네트워크 운영체제이다. RETOS에 운영체제의 일부 기능을 모듈화 하여 탑재 및 변경할 수 있도록 응용프로그램과 커널 모듈 그리고 커널을 계층화 하였다.

• 본 연구는 한국과학재단에서 지원하는 국가지정연구실사업으로 수행하였음 (과제번호 : 2005-01352)

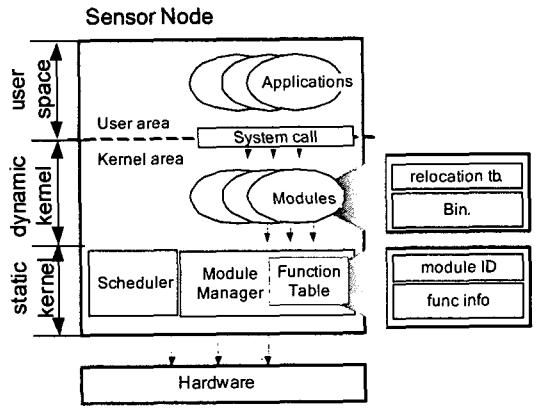


그림 1 RETOS 커널

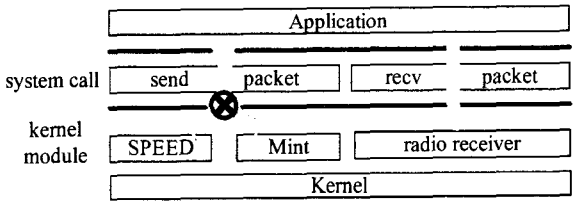


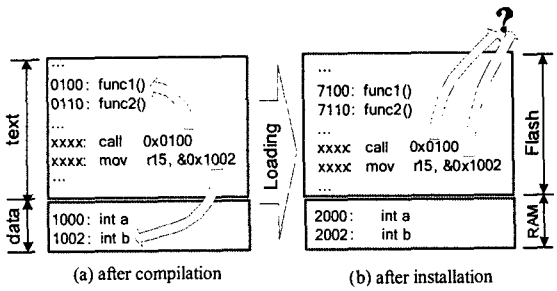
그림 2 다양한 커널 모듈의 지원

2. 동적 커널 모듈 지원 기법

RETOS는 응용 프로그램의 수행에 있어 멀티 쓰레딩을 지원하는 센서 네트워크 범용 운영체제이다. RETOS에는 다양한 어플리케이션이 실행되며 실시간으로 응용프로그램의 수행 및 정지, 교체 등이 가능하다. 운영체제에서 작동하는 다양한 응용프로그램은 커널의 다양한 기능을 사용한다. 이런 다양한 기능을 제공하기 위해 커널의 각 기능을 모듈화 하여 구현할 수 있도록 동적 커널 시스템을 RETOS에 구현하였다. 그림1은 동적 커널 모듈이 지원 되는 RETOS 커널의 전체적인 모습이다. 시스템은 크게 커널과 응용프로그램으로 나누어진다. 커널은 커널 모듈을 동적으로 설치하고 해제할 수 있도록 커널 모듈 매니저를 사용한다. 커널의 설치는 동적 링킹 과정으로 Relocation 방식을 이용하며, 커널 모듈 간 함수 연결을 위해 함수 테이블을 유지, 관리한다. 설치가 끝나면 커널과 커널 모듈은 하나의 바이너리처럼 연결되어 동작한다. 한편, 응용프로그램은 스케줄러에 의해 스케줄 되어 수행되며, 시스템 콜 함수를 통해 커널의 서비스를 이용한다. 커널은 사용하는 하드웨어에 따라 또는 기능 구현 방법에 따라 또는 배터리 잔량 등 노드의 상태에 따라 다양한 구현의 커널 모듈을 가지고 있다. 응용프로그램의 시스템 콜은 상황에 따라 적합한 커널 모듈을 선택적으로 사용할 수 있어 상황에 맞는 커널 기능을 실행하게 된다. 그림 2는 이 같은 커널 모듈 선택을 설명한다. 응용프로그램은 패킷을 다른 노드로 보내기 위해 SendPacket 함수를 호출하지만 실제로 그 패킷은 다양한 종류의 라우팅 기법을 통해 전송될 수 있다.

2.1 동적 모듈 링킹

일반적으로 센서 네트워크에 사용되는 하드웨어는 MMU가 없는 MCU를 사용한다. MMU는 물리 메모리 공간과 가상 메모리 공간을 서로 변환해 주고 메모리 접근 권한을 제어해 주는 역할을 해 준다. 프로그램의 코드가 TinyOS[1]처럼 싱글 이미지인 경우, 코드는 RAM 상에 항상 같은 위치에 로딩 되게 할 수 있기 때문에 MMU가 없는 것은 문제가 되지 않는다. 하지만, 바이너리가 여러 개로 관리되는 경우 컴파일 타임에 바이너리 이미지가 메모리의 어느 곳에 로딩 될 지 보장 받을 수 없으므로 함수 콜이나 변수 접근에 있어 주소 처리에 문제가 발생한다. 그림 3의 (a)는 임의의 코드를 Stack Base와 Text Area 의 주소를 임의로 정의하여 컴파일 한 예이다. 해당 코드는 주소 100 의 위치에 있는 func1 과 1002 에 있는 var 2를 접근한다. (b) 는 이 컴파일 된 바이너리 이미지가 data는 2000 번지에 코드는 플래시 메모리 상의 10000 번지에 저장된 경우이다. 만약, 바이너리 이미지가 수정 없이 로딩 된다면 잘 못 된 곳을 참고 하게 된다. 이런 문제를 해결하기 위한 방법으로 Module 의 모든 메모리 참조를 알맞은 주소로 치환하는 Relocation 방법을 사용하였다. Relocation은 코드 컴파일 시 미리 정의한 메모리주소를 기준으로 컴파일하고, 코드 내부에서 사용되는 메모리 참조를 Relocation Table에 저장해 두었다가 메모리 로딩 시 메모리 참조를 올바른 값으로 변경해 주는 방법이다. 이 경우 모듈 설치 시 Relocation 절차에 따른 오버헤드가 있다는 단점이 있지만 모든 메모리 접근이 절대 주소를 통해 이루어지므로 실행 성능상의 이점이 있다. 한편 SOS는 코드 내에 절대 주소가 없도록 Position Independent Code를 작성한다. PIC 는 로딩 된 메모리 위치에 관계없이 실행 할 수 있다는 장점은 있으나 Indirect 메모리 접근으로 인한 오버헤드를 가진다.



(a) after compilation (b) after installation

그림 3 링킹 과정에서 잘못된 주소를 참조하는 문제

uClinux의 Relocation 은 플래시 메모리에 저장된 바이너리 코드를 RAM 으로 로딩하면서 실행된다. 따라서 플래시 메모리 상의 바이너리는 잘못 된 주소를 참조하는 코드를 포함하고 있고, 플래시 메모리 상에서 바로 실행할 수 없다. 하지만, RETOS 의 경우 코드를 Flash 에 저장하면서 RAM 상의 relocation 과 함께 플래시 메모리 상의 relocation 도 수행한다. 이를 통해 Flash 상에 유효한 바이너리를 저장하게 되었고 플래시 상에서 코드를 바로 실행할 수 있다.

2.2 메모리 절약 기법

바이너리 코드는 실행을 위해 코드 영역, 스택 영역, 글로벌 변수 영역, 동적 메모리 영역을 사용한다. 최근 개발 된 다수의 임베디드 시스템은 플래시 메모리 상에서 직접 코드를 실행하는 eXecute In Place (XIP)를 지원하므로 메모리상의 코드 영역 할당은 하지 않는다. 동적 메모리 할당은 커널 모듈의 필요에 의해 그 크기가 정해지므로 그 크기를 임의로 줄 일 수 없다. 따라서 이 섹션에서는 스택 영역과 글로벌 변수 영역에 대한 최적화 방법을 제안한다.

스택 영역은 그 사용량을 컴파일 타임에 알 수 없다. 커널 모듈을 위해 스택의 크기를 충분한 만큼 설정하더라도 모듈 마다 고정 크기의 메모리를 할당하여 주어야 하므로 메모리 낭비를 초래한다. 이를 최소화하기 위해 커널 모듈이 코드 실행을 위해 하나의 공통 스택을 사용하도록 하였다. 모든 커널 모듈은 같은 스택을 이용하여 모듈 함수를 실행하게 되고 그 만큼 메모리의 절약을 가져온다. 하지만, 커널 모듈 코드 실행 중간에 컨텍스트가 스위칭 되어 다른 모듈 함수가 실행 되게 되면 커널 스택 상의 데이터의 안전을 보장할 수 없으므로 모듈 함수 도중 컨텍스트가 스위칭 되는 것을 막았다.

2.3 커널 안정성

RETOS는 Dual Mode Operation 을 제공한다. 범용 운영체제에서 Dual Mode Operation은 하드웨어를 통해 지원되지만 RETOS는 탑재되는 하드웨어의 제약으로 소프트웨어적으로 응용프로그램은 유저 모드로, 커널 모듈은 커널 모드로 동작하게하고 서로 다른 메모리 영역을 사용하도록 하였다. 응용프로그램은 커널이 제공하는 시스템 콜을 통해 커널 모듈의 함수를 실행하게 되는데, 이때, 코드 Jump 와 함께 커널 모드로의 Mode 전환, 커널 스택으로의 Stack 전환 등의 작업을 하게 된다. 커널 모듈의 코드 위치가 유저 코드와 다르므로 시스템 콜을 거치지 않고서는 유저 코드는 커널 모듈의 코드로 점프할 수 없다. 따라서 응용프로그램이 커널 모듈의 함수를 잘못된 방법으로 사용할 수 없게 된다. 또한 커널 모듈은 커널과 함께 응용프로그램과 독립된 메모리 영역을 사용하기 때문에 응용프로그램의 잘못 된 주소 참조로 인한 커널 패닉을 미리 방지한다.

또한, 앞서 언급했듯이 커널 모듈은 스택 영역의 절약을 위해 커널의 스택을 공유하여 사용한다. 이것은 유저 응용프로그램의 스택과 분리되어 있으므로 유저 응용프로그램의 오류나 공격으로 인해 커널 모듈의 스택 내 데이터를 가져 못하도록 하는 이점을 준다. 또한 모든 커널 모듈은 하나의 공통 스택 영역을 사용하기 때문에, 한 번에 한 개의 모듈만 동작한다. 따라서 서로 다른 모듈이 서로의 스택상의 데이터를 임의로 가져올 수 없다.

Table 1. RETOS LKM Cost

작업	Cost
Total Compilation Time	1.332 sec
Compile Overhead	0.270 sec
Sample Module Size	1714 bytes
Relocation Table Size	213 bytes
Relocation Processing Time (Linking)	0.1 sec 이하
Execution Overhead	0

3. 시스템 구현

Loadable Kernel 을 구현하기 위한 플랫폼으로 RETOS version 0.86을 이용하였다. 0.86 버전은 멀티 쓰레드 응용 프로그램을 탑재할 수 있도록 되어 있으며, UART 케이블을 통해 OS 커널과 커널 모듈, 응용 프로그램을 로딩 할 수 있다. 커널과 커널 모듈 그리고 응용 프로그램은 모두 분리된 코드로 존재하며, 응용 프로그램은 RETOS 스케줄러에 의해 스케줄링되어 실행된다.

RETOS 시스템에 커널 모듈 시스템이 들어가면서 발생하는 오버헤드를 Tree Routing Module을 제작하여 측정하였다. 커널 모듈 시스템이 가지는 오버헤드는 컴파일, 배포, 설치, 실행 과정을 통해 분석해 볼 수 있다. 커널이 모듈 단위로 구성되어 기존의 커널에 동적으로 설치하기 위해 Relocation Table 을 생성하였다. 이는 컴파일 시 기존의 코드에서 메모리 접근을 추출해 내어 테이블로 유지하는 과정이 포함된다. 이 과정은 바이너리 파일을 한번 스캔함으로써 얻어 낼 수 있다. 실제 측정 결과는 Table 1을 통해 Relocation 수행 시간이 전체 컴파일 수행 시간의 20%를 차지함을 보여준다. 이 오버헤드는 크로스 컴파일러가 존재하는 쉘의 인터프리팅 과정에서 오는 오버헤드가 대부분으로 추후 컴파일 과정으로 포함하면 그 오버헤드를 최소화 할 수 있다. 컴파일 된 모듈은 네트워크를 통해 배포할 때 그 패킷에 위 과정에서 생성된 Relocation Table 을 포함하게 된다. 이 사이즈는 초기 샘플 모듈의 사이즈가 1714 bytes 일 때, 213 bytes 로 약 11%의 네트워크 패킷 증가를 초래한다. 커널 모듈은 기존의 커널에 삽입되기 위해 대상 노드에서 다운로드 과정을 거친 후 Linking 과정을 거치게 된다. 이 때 Relocation Table 에 등록 되어 있는 모든 메모리 접근의 주소를 올바른 주소로 바꿔주는 작업을 거치게 된다. 이 과정에서 수행 시간은 약간 증가하였지만 이는 매우 순수 CPU 작업이므로 플래시 쓰기 작업에 비해 매우 미비하다. 한편, 설치된 모듈은 모든 함수 호출 및 메모리 참조를 다이렉트 방식으로 접근하게 되므로 하나의 바이너리 코드로 구성되어있는 것과 동일한 수행 속도를 보여준다.

4. 결론

RETOS는 센서네트워크를 위한 범용 운영체제이다. RETOS 는 다양한 센서 응용을 지원하기 위해 커널의 구성을 상황에 따라 수정할 수 있도록 커널을 모듈화 하여 제작하고 동적으로 삽입하는 메커니즘을 제공한다. 지금까지의 센서 네트워크 용 운영체제는 개발의 용이함과 재배포를 위해 시스템 전체를 새로 재배포하거나 응용 프로그램을 모듈화 하여 작성하여 재배포하는 방식으로 개발되었으나, RETOS는 커널 자체를 모듈화 하여 상황에 따라 선택적으로 재구성할 수 있는 메커니즘을 제공하고 있어 좀 더 응용 프로그램에 독립적이고 추상적인 시스템 최적화를 가능하게 한다. 이런 특징은 시스템 버그 수정이나 시스템 기능 추가 시 이미 배포된 센서 네트워크에 동적으로 업데이트를 할 수 있게 하며, 응용에 따른 다른 커널 구성으로 보다 정밀한 동작과 자원 절약적인 수행을 가능하게 한다. RETOS는 적응형 시스템을 향한 진보 된 모습의 운영체제이다. 모듈화 된 모듈의 배포 과정에서 발생하는 프로토콜 문제나 커널 최적화 방법은 향후 과제로 남긴다.

참고 문헌

[1] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister, "System architecture directions for network sensors," Proceedings of the ASPLOS-IX, Cambridge, Massachusetts, USA, November 2000.

[2] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler and Mani Srivastava, "A Dynamic Operating System for Sensor Nodes," Proceedings of the 3rd international conference on Mobile systems, applications, and services, Seattle, Washington, USA, 2005.

[3] Philip Levis and David Culler, "Maté: A Tiny Virtual Machine for Sensor Networks," Proceedings of the ASPLOS-X , San Jose, California, USA, 2002.

[4] Hyoseung Kim and Hojung Cha, "Towards a Reliable Operating System for Wireless Sensor Networks," The USENIX Annual Technical Conference: Systems Practice & Experience Track, Boston, Massachusetts, USA, May 2006.

[5] Rimon Barr, John C. Bicket, Daniel S. Dantas, Bowei Du, T. W. Danny Kim, Bing Zhou, Emin Gün Sirer, "On the need for system-level support for ad hoc and sensor networks," ACM SIGOPS Operating Systems Review, vol.36 no.2, pp.1-5, April 2002.

[6] A.Dunkels, B. Gronvall and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, Tampa, Florida, USA, 2004.

[7] Joel Koshy and Raju Pandey, "VM★: Synthesizing Scalable Runtime Environments for Sensor Networks," Proceedings of the 3rd international conference on Embedded networked sensor systems, San Diego, California, USA, 2005.