공간 제약하의 센서 운영체제를 위한 동적 쓰레드 스택관리 기법

(Dynamic Threads Stack Management Scheme for Sensor Operating Systems under Space-Constrained)

이 상 호 † 조 유 근 †† 홍 지 만 †††
(Sangho Yi) (Yookun Cho) (Jiman Hong)

요 약 무선 센서 네트워크는 자연 환경의 정보를 수집하고, 수집한 정보를 가공하고, 가공된 정보를 무선 통신을 통하여 사용자에게 실시간으로 전달하는 기능을 가진 설비이다. 이러한 센서 네트워크는 다수의 무선 센서 노드들로 이루어지고, 이 센서 노드들은 비용 효율성의 이유로 매우 제한적인 하드웨어 칩들로 구성된다. 예를 들어, UC Berkeley에서 설계한 MICA센서 노드에는 8-bit CPU, 4KB RAM, 그리고, 128KB FLASH 등으로 구성된다. 따라서 이것들을 동작시키는 센서 운영체제는 이러한 하드웨어 제약성을 감내할 수 있어야 한다. 본 논문에서는 멀티 쓰레디드 센서 운영체제를 위한 공간 효율적인 쓰레드 스택 관리 기법을 제안한다. 제안한 기법은 컴파일 시점에 각 쓰레드 함수의 스택 사용량 정보를 측정한다. 측정된 결과를 바탕으로, 함수 호출 시와 같은 스택 영역의 요구가 발생할 경우에 스택의 할당 및 반환 작업을 수행하여 쓰레드 스택 영역을 동적으로 관리한다. 본 기법은 나노 Qplus 센서 운영체제에서 구현되었다. 본 논문의 성능 실험을 통하여, 제안한 기법을 사용하는 것이 기존의 정적인 스택 관리 방법을 사용하는 것 보다 스택 메모리 공간을 보다 효율적으로 관리할 수 있음을 확인한다.

키워드 : 쓰레드 스택 관리, 멀티 쓰레딩, 무선 센서 네트워크, 센서 운영체제

Abstract Wireless sensor networks are sensing, computing and communication infrastructures that allow us to monitor, instrument, observe, and respond to phenomena in the harsh environment. Generally, the wireless sensor networks are composed of many deployed sensor nodes that were designed to be very cost-efficient in terms of production cost. For example, UC Berkeley's MICA motes have only 8-bit CPU, 4KB RAM, and 128KB FLASH memory space. Therefore, sensor operating systems that run on the sensor nodes should be able to operate efficiently in terms of the resource management. In this paper, we present a dynamic threads stack management scheme for space-constrained and multi-threaded sensor operating systems. In this scheme, the necessary stack space of each function is measured on compile-time. Then, the information is used to dynamically allocate and release each function's stack space on run-time. It was implemented in Nano-Qplus sensor operating system. Our experimental results show that the proposed scheme outperforms the existing fixed-size stack allocation mechanism.

Key words : Thread Stack Management, Multi-threading, Wireless Sensor Networks, Sensor Operating Systems, Memory Management

· 본 연구는 서울대학교 BK21과 컴퓨터 연구소의 지원으로 이루어졌음
본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음

† 학생회원 : 서울대학교 컴퓨터공학부
shyi@os.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
cho@os.snu.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학부 교수
jiman@ssu.ac.kr
(Corresponding author임)

논문접수 : 2007년 4월 27일
심사완료 : 2007년 8월 24일

: 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지: 시스템 및 이론 제34권 제11호(2007.12)
Copyright@2007 한국정보과학회

1. 서 론

오늘날, 컴퓨터 시스템 분야에서 다양한 운영체제들이 존재하고 있다. 이러한 운영체제들은 동작 방식 및 커널의 내부 구조에 따라서 크게 두 가지 시스템 모델로 분류할 수 있다. 역사적으로, 이 두 가지 모델의 이름은 다음과 같이 명명되었다: 이벤트 드리븐(혹은 메시지 지향) 시스템과 멀티 쓰레디드(혹은 프로시저 지향) 시스템. 지난 컴퓨터 과학 분야의 다양한 연구에서 위의 두 가지 시스템 모델에 대한 비교 분석 및 토론이 이루어져왔다[1-6]. 1978년, Lauer와 Needham은 이러한 논쟁을 종식시키고자 하였다[2]. 그 결과로, 이들은 위의 두 가지 시스템 모델은 서로 다른 모습을 갖지만 동등한 "듀얼(Dual)"임을 주장하였고, 기능 및 성능상의 영향을 줄 수 있는 다양한 예제들을 바탕으로 이를 뒷받침하였다. 이론상으로, 위의 결론은 정확했다. 그러나 실제의 구현에 있어서는 시스템의 효율성 및 성능의 측면에서 상당한 차이를 보이고 있다. 이후의 연구인 [3-6]에서는, 각 시스템의 특성에 따라서 이벤트 드리븐 방식과 멀티 쓰레디드 방식 중 무엇을 사용하는 것이 시스템의 효율성 및 성능에 좋은지 언급하였다. 이벤트 드리븐 방식과 멀티 쓰레디드 방식의 장단점은 다음과 같다. 이벤트 드리븐 시스템 설계 방식은 태스크 문맥 교환 시간이 상대적으로 짧게 처리되고, 메모리 공간을 적게 사용하는 장점을 가진다. 그러나 선점이 불가능하며, 각 이벤트 처리기가 수행 종료되기 전까지는 다른 이벤트를 처리해줄 수 없으므로, 이벤트의 처리 응답 시간이 어느 정도가 될지 미리 알 수 없게 된다. 그와 반대로, 멀티 쓰레디드 시스템 설계 방식은 태스크 문맥 교환이 상대적으로 오랜 시간을 소모하고, 메모리를 좀 더 사용하는 단점을 지닌다. 하지만, 어느 시점에서도 선점이 가능하며, 이를 통하여 각 태스크 별로 동일한 타임 슬라이스를 할당 받을 수 있다. 따라서 하나의 작업에 대한 처리 응답 시간이 결정적이므로 실시간 시스템에서 사용하기에 더 나은 방법이다.

위의 두 가지 모델을 따르는 운영체제들은 쉽게 찾아볼 수 있다. 예를 들면, 리눅스와 Windows XP는 전형적인 멀티 쓰레디드 운영체제들이고, Symbian OS, Parm OS, 그리고 Windows CE는 이벤트 드리븐 운영체제의 예이다. 이러한 예는 무선 센서 네트워크를 위하여 설계 및 구현된 센서 운영체제들에서도 다양하게 찾아볼 수 있다[7-14]. TinyOS[7], SOS[8], PEEROS[9], 그리고 CORMOS[10] 등은 이벤트 드리븐 운영체제이고, 반대로 MANTIS OS[11], Nano-Qplus[12], 그리고 RETOS[13] 등은 멀티 쓰레디드 운영체제의 예이다. 또한, Contiki[14]는 이벤트 드리븐 커널 위에서 멀티 쓰레디드 응용을 수행시킬 수 있는 하이브리드 형태의 운영체제이다. 이러한 센서 운영체제들은 상당한 하드웨어 제약요소를 갖는 무선 센서 노드에서 동작한다. 한 예로, 버클리 대학에서 제작한 MICA 계열의 센서 노드는 8비트의 AVR 마이크로프로세서와 4KB 정도의 RAM 공간을 가지고, 2개의 AA 배터리로 동작한다[15]. 센서 운영체제는 이러한 제약하에서 동작하며, 센싱, 데이타 수집, 멀티 홉 무선 통신 등의 기능을 실시간적으로 처리해줄 수 있어야 한다.

본 논문에서는, 센서 노드에서 동작하는 멀티 쓰레디드 센서 운영체제를 위한 공간 효율적인 동적 쓰레드 스택 관리 기법을 제안한다. 제안한 기법을 사용하면 기존의 센서 운영체제에서 사용되고 있는 고정 크기 스택 할당 기법을 사용하는 것 보다 메모리 단편화 문제를 상당히 절감시킬 수 있다. 제안한 기법은 컴파일 시점에 각 쓰레드 함수별 스택 사용량을 측정한다. 측정된 결과를 바탕으로, 함수 호출 시에 필요로 하는 크기의 스택 영역을 할당하고, 함수 종료 시에 반환하여 동적으로 쓰레드의 스택을 관리할 수 있게 한다. 본 논문의 비교 실험을 통하여, 제안한 기법을 사용하는 것이 기존의 고정 크기의 스택 관리 방법을 사용하는 것 보다 메모리 효율성을 상당히 증진시킬 수 있고, 전체 수행 시간도 나빠지지 않음을 보인다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 보이고, 3절에서는 공간 제약하의 멀티 쓰레디드 센서 운영체제를 위한 동적 스택 영역 관리 기법의 자세한 설명과 함께 그 구현 방법을 보인다. 4절에서는 제안한 기법의 성능 측정 결과를 보이고 이를 평가한다. 마지막으로, 5절에서 본 논문의 결론을 보인다.

2. 관련 연구

본 절에서는 동적 스택 관리 기법 및 멀티 쓰레디드 설계 방식에서의 스택 관리와 관련하여 기존의 여러 가지 관련 연구를 보인다. 현재까지, 동적 스택 관리의 문제와 관련하여 스택 메모리 공간의 관리를 효율적으로 수행하기 위한 몇 가지의 기법들이 연구되어왔다[4-6].

Adya 등은 [4]에서, 멀티 쓰레디드 시스템과 이벤트 드리븐 시스템의 장점을 융합한 형태인 "sweet spot"을 제시하였다. 이 방법을 사용하여, 멀티 쓰레디드 방식의 단점인 공간 낭비 문제를 감소시킬 수 있고, 이벤트 드리븐의 상대적 단점인 비선점성의 문제를 완화할 수 있음을 언급하였다.

Ousterhout는 [3]에서, 멀티 쓰레디드 태스크 모델의 단점을 다양한 응용 예제들을 바탕으로 언급하였다. 그 결과로, 시스템이 성능의 측면에서 응답 시간의 안정성

이나 선점 가능성에 큰 영향을 받을 경우에는, 멀티 쓰레디드 방식을 도입하는 것이 이벤트 드리븐 방식보다 더 나음을 보였다. 본 논문에서는, 센서 네트워크의 실시간 응답 시간의 보장을 위하여 멀티 쓰레디드 센서 운영체제 기반에서 작업을 수행하였다.

Behren 등은 [5]에서, 멀티 쓰레디드 방식과 이벤트 드리븐 방식의 성능 비교를 수행하였다. 그 결과로, 멀티 쓰레디드 방식의 경우에는 컴파일러 지원 기법을 통하여 메모리 사용량 및 수행 시간의 측면에서 성능이 향상될 수 있음을 확인하였다. 본 논문에서는, 컴파일 타임의 분석 기법을 활용하여, 각 쓰레드 함수들의 스택 필요량을 미리 계산하는 방식으로 메모리 사용량의 측면에서 효율성 및 성능을 향상시켰다.

Torgerson은 [9]에서 각 쓰레드의 최대 스택 사용량을 측정하여, 각 쓰레드의 초기 스택 할당 시에 이 정보를 사용하게 하는 스택 관리 기법을 제안하였다. 이를 통하여 스택 오버플로우 문제를 제거하였다. 그러나 런타임에 스택을 동적으로 관리하는 기능은 제공하지 못하였고, 따라서 공간의 효율성은 그다지 증진시키지 못하였다.

Tismer는 [4]에서, 파이썬 인터프리터의 확장성 증진을 위한 스택리스 파이썬을 제안하였다. 이 방법을 통하여, 기존의 정적 스택 할당 방법을 사용한 경우보다 파이썬 인터프리터가 보다 안정적으로 확장 가능하게 동작할 수 있었다. 이 논문은 스택리스와 유사한 방식의 설계를 갖는 본 논문의 배경 지식이 되었다.

Gustafsson은 [6]에서, C언어의 특징에 관하여 언급하면서, 쓰레드의 스택을 공간 효율적으로 구성하기 위한 몇 가지 기법을 소개하였다. 그러나 실제로 구현한 예는 이 논문에서 찾아볼 수 없었다. 본 논문에서는 [6]의 논문에서 소개된 기법을 실제로 센서 운영체제에 접목시켜 구현하였고, 그 결과 공간 효율적인 쓰레드 스택 관리가 가능함을 확인하였다.

Dunkel 등은 [6]에서, C언어에서의 스택리스 쓰레드 프로그래밍 방법을 제안하였다. 이 기법은 멀티 쓰레드 프로그래밍을 단 하나의 스택에서 동작시키는 프로그래밍 기법이었다. 그러나 그들이 제안한 방법은 이벤트 기반 프로그래밍의 한 변형이었고, 멀티 쓰레드의 장점인 선점 기능을 지원하지 않았다.

3. 멀티 쓰레디드 센서 운영체제를 위한 동적 스택 관리 기법의 설계 및 구현

본 절에서는 무선 센서 네트워크 및 무선 센서 플랫폼의 요구사항과 제약사항을 보인다. 이러한 제약사항 및 요구사항 등을 토대로 본 논문의 핵심인 멀티 쓰레디드 센서 운영체제를 위한 공간 효율적인 동적 스택 영역 관리 기법의 설계 및 구현 내용을 자세히 설명한다.

3.1 무선 센서 네트워크의 요구사항

일반적으로 무선 센서 네트워크는 다수의 센서 노드로 구성되고, 비용 효율성 때문에 각 센서 노드는 매우 제한적인 하드웨어들로 구성된다. 대표적인 예로, 미국의 버클리 대학에서 설계한 MICA 시리즈 센서 노드는 8비트의 저전력 마이크로프로세서, 4KB 크기의RAM 공간, 그리고 2개의 AA 배터리 등으로 구성된다[18]. 이러한 센서 플랫폼에서 동작할 센서 운영체제는 제한적인 메모리 공간을 효율적으로 관리하면서 제한적인 배터리 전력을 효율적으로 사용해야한다. 또한, 센서 운영체제는 다양한 형태의 작업(예: 센싱, 데이타 수집 및 병합, 멀티 홉 무선 통신)을 실시간적으로 처리할 수 있어야 한다. 이러한 이유로, 멀티 쓰레디드 센서 운영체제와 관련한 연구가 최근에 많은 대학에서 수행되고 있고[11-13], 멀티 쓰레디드 센서 운영체제에서는 공간 효율적인 쓰레드 스택 관리 기법이 필요하다. 다음은 센서 운영체제를 위한 스택 관리 관리 기법의 설계 및 구현 시에 반드시 고려해야 할 무선 센서 네트워크 및 센서 노드의 몇 가지 요구사항들이다.

- 스택 할당량의 최소화: 센서 노드의 전체 메모리 공간은 약 4KB 정도로 매우 제한적이다. 따라서 쓰레드를 위해 할당되는 스택 영역의 크기는 최소한의 크기가 되어야 메모리 공간의 부족 문제를 완화할 수 있을 것이다.
- 시간상 오버헤드의 최소화: 무선 센서 플랫폼에서의 수행 시간상의 오버헤드는 에너지 사용량 및 노드의 수명과 밀접한 관련이 있다. 따라서 동적 스택 할당 및 반환 작업의 시간상 오버헤드를 최소화 한다면, 제한적인 배터리를 보다 효율적으로 사용할 수 있을 것이다.

3.2 동적 스택 관리 기법의 설계 및 구현

일반적으로, 멀티 쓰레디드 태스크 모델에서 각 태스크는 쓰레드로 표시되며, 쓰레드들은 코드와 데이타 자원을 공유하며 고유의 스택 공간과 레지스터 셋을 가진다. 현존하는 모든 멀티 쓰레디드 센서 운영체제에서는, 각 쓰레드의 스택 공간은 정적으로 쓰레드의 초기 생성 시에 할당되고, 쓰레드의 종료 시에 반환되는 방식으로 동작하였다. 이 경우, 각 쓰레드가 실제로 할당된 스택 공간을 모두 사용하지 않더라도 정적으로 할당되어 버리기 때문에 공간상의 효율성이 감소하게 된다.

본 논문에서 제안하는 동적 스택 관리 기법의 주 목적은 앞서 보인 공간상의 비효율성을 완화하여 보다 효율적으로 동작하게 하는 것에 있다. 이를 위하여, 동적으로 쓰레드의 스택을 관리하여, 필요로 하는 크기만큼의 공간을 할당 혹은 반환하게 처리한다. 이 기법에서,

필요로 하는 스택 영역의 크기는 어셈블리 언어로 변환된 소스코드를 분석하여 알아낼 수 있다.

다음의 표 1은 기존의 정적 스택 관리 기법과 제안한 동적 스택 관리 기법의 기능상의 비교를 보인다. 정적 스택 관리 기법에서, 쓰레드 스택의 할당 및 반환은 단지 쓰레드의 생성 및 제거 시점에서만 발생한다. 반면에 제안한 동적 관리 기법에서는 각 함수 호출 시점 및 함수 종료 시점에 이러한 일이 발생할 수 있다. 따라서 제안한 기법에서의 쓰레드 스택의 크기는 동적으로 변화하게 되며, 이로 인한 시간상의 오버헤드도 발생할 수 있다. 기존의 기법에서는 정적인 스택 크기를 갖게 되므로, 스택을 넘쳐서 사용했을 경우, 다른 쓰레드를 침범하는 스택 오버플로우가 발생할 수 있고, 이로 인하여 시스템이 붕괴될 수 있다. 그러나 제안한 기법에서는 스택의 사용량을 바탕으로 동적으로 관리하므로, 스택 오버플로우 문제는 전혀 발생하지 않는다.

다음의 그림 1은 나노 Qplus 운영체제에 존재하는 응용 프로그램 예제인 "pthread_net4:sensor110"의 수행 시에 실제로 사용하는 스택 영역의 크기를 출력한 것이다. 이 그림의 결과를 바탕으로, 쓰레드의 스택 사용량은 변화가 상당히 심한 것을 볼 수 있다. 또한, 정적인 크기로 스택을 할당할 경우에는 전체 메모리 공간의 효율성이 크게 감소함을 확인할 수 있다. 이 예제는, 초기 실행 시에만 많은 스택 영역을 사용할 뿐, 그 외의 경우에는 상대적으로 작은 크기의 스택 영역만을 필요로 하고 있다. 따라서, 동적인 스택 할당을 수행한다면, 스택 사용량의 변화에 맞추어 스택을 효율적으로 할당할 수 있을 것이다.

그림 2는 제안한 동적 스택 관리 기법을 사용하였을 때의 컴파일 과정을 보인다. 제안한 기법은 크게 다음의 두 가지 모듈로 구성된다, 첫째로, "스택 사용량 측정기"가 존재한다. 그리고 이 측정기를 통하여 얻은 결과를 사용하여 동적 스택 할당 및 반환 코드를 삽입하는 "동적 스택 할당 및 반환 코드 삽입기"가 존재한다.

먼저, C 소스 코드를 AVR-GCC 컴파일러[20]의 "-S" 옵션을 사용하여 컴파일을 수행하면 출력 결과로 어셈블리 코드 파일이 나타난다. 이 파일을 "스택 사용량 측정기"를 통하여 각 함수별 스택 사용량을 측정한다. 이 결과를 바탕으로, "동적 스택 할당 및 반환 코드 삽입기"를 사용하여 원본 C 소스 코드를 수정한다. 그 결과로 수정된 파일을 AVR-GCC 컴파일러를 사용하여 컴파일 과정을 수행하면 최종적으로 타겟 이미지가 작성된다.

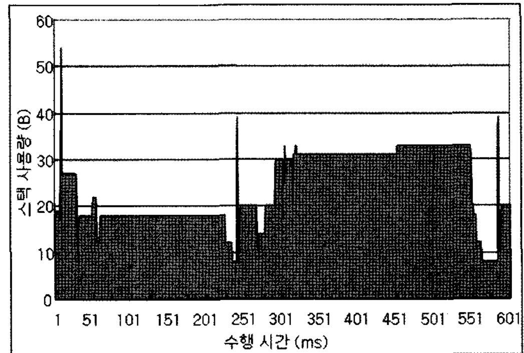


그림 1 "pthread_net4:sensor110"의 스택 사용량

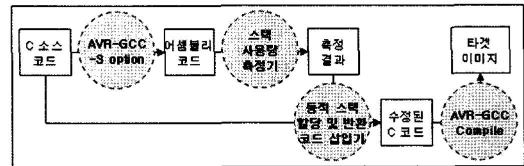


그림 2 제안한 기법을 사용하였을 때의 컴파일 과정

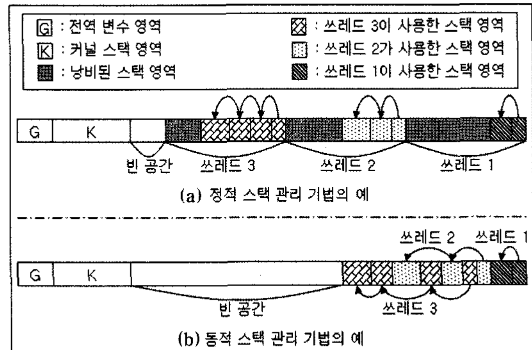


그림 3 기존의 기법과 제안한 기법의 스택 사용 예

위의 그림 3은 기존의 정적 스택 관리 기법과 제안한 동적 스택 관리 기법을 각각 사용하였을 때의 전체 메모리 맵을 나타낸 것이다. 기존의 정적 관리 기법에서는

표 1 기존의 기법과 제안한 기법의 기능상의 비교

기능	정적 스택 관리	동적 스택 관리
스택 할당	쓰레드 생성 시	쓰레드 내의 함수 콜 시
스택 반환	쓰레드 종료 시	쓰레드 내의 함수 리턴 시
스택 크기	고정 크기	가변적으로 변함
시간상 오버헤드	쓰레드 생성 및 종료 시의 할당/반환 비용발생	함수 콜 및 리턴 수만큼의 할당/반환 비용발생
스택 오버플로우	고정 스택 크기를 넘어설 경우 발생	필요한 만큼 미리 할당하므로, 발생하지 않음

쓰레드 공간을 정적으로 할당하므로, 공간의 낭비가 발생하는 모습을 보여주고 있다. 그러나 동적인 스택 관리 기법에서는 필요한 공간만을 힙 영역에서 동적으로 할당하므로 자유 공간이 늘어난 모습을 보이고 있다. 따라서 제안한 기법을 사용하면 보다 많은 쓰레드를 보다 효율적으로 동시에 생성할 수 있게 된다.

다음의 표 2에는 본 논문에서 제안한 기법의 동작 방식 및 알고리즘 설명을 위하여 사용한 기호 및 함수의 정의를 나타내었다.

표 2 본 논문에서 사용한 기호 및 함수의 정의

기호 및 함수	세부 설명
spaceResidual	현 스택 공간에서의 스택 영역 잔량
spaceAlpha	예측 관리 기법에서의 할당 기본 단위
memAlloc(size)	"size" 만큼의 스택을 할당
memFree(addr)	"addr"이 가리키는 스택을 반환
saveSP(addr)	"addr"에 스택 포인터를 저장
loadSP(addr)	"addr"의 값으로 스택 포인터를 복원
changeSP(value)	스택 포인터를 "value"로 변경
push(value)	"value"를 스택에 푸쉬함
call(func())	"func()"를 호출함
big(value1,value2)	"value1"과 "value2" 중에서 큰 값을 선택
func()	C 언어 수준의 함수
func().addrStack	함수의 스택 주소
func().spaceRequired	함수의 스택 필요량 (측정된 값)
func().arguments	함수의 인자값들

위의 기호 및 함수의 정의를 바탕으로, 제안한 기법의 동작 방식을 아래의 알고리즘 1에 나타내었다.

알고리즘 1. 스택 영역 할당, 함수 호출, 반납 과정

- 쓰레드가 sunny() 함수를 호출 할 때, 다음의 과정을 수행 -

```
1 sunny().addrStack = memAlloc(sunny().
  spaceRequired)
2 saveSP(sunny().addrStack)
3 changeSP(sunny().addrStack+sunny().
  spaceRequired -1)
4 push(sunny().arguments)
5 call(sunny())
6 loadSP(sunny().addrStack)
7 memFree(sunny().addrStack)
```

위의 알고리즘 1에서, 스택 할당 및 스택 포인터 저장은 함수 호출 이전에 일어나고, 함수가 리턴 된 후에 스택 포인터를 복원하고 할당했던 스택을 반환한다. 이러한 방식으로 동작하므로, 각 쓰레드 스택은 필요로 하는 양만이 할당되고 반환된다.

3.3 예측 버퍼를 사용하는 동적 스택 관리 기법

앞에서 제안한 기법은 각 함수의 스택 필요량에 따라 스택을 할당하고 반환하는 정책을 사용하였다. 이러한 방식을 도입하면 전체 메모리 공간의 효율성은 극대화 될 수 있지만, 여러 번의 메모리 할당 및 반환 작업을 수행해야 하므로 수행 시간의 측면에서 오버헤드가 생기게 된다. 이 문제를 완화하기 위하여, 예측 버퍼(Look-ahead buffer)를 사용하여 스택을 동적으로 할당 및 반환하는 방법을 생각해볼 수 있다. 이 방법은 표 2에 보인 "할당기본단위"를 사용하여, 함수가 필요로 하는 스택을 할당할 때에, 최소 단위로 이 값을 사용하게 된다. 이 방법을 사용하면, 기존의 정적 할당 기법과 제안한 동적 할당 기법의 하이브리드 형태로 동작하여, 수행시간 상의 오버헤드를 감소시킬 수 있다. 다음의 그림 4는 예측 버퍼를 사용하는 기법과 앞에서 보인 일반적인 동적 스택 관리 기법의 비교를 보인다.

그림 4에서, 예측 버퍼에 기반한 동적 스택 관리 기법은 일반적인 동적 관리 기법에 비하여 메모리 할당 연산이 적게 일어나는 것을 확인할 수 있다. 이 그림에서는 "할당기본단위"가 각각 5, 10, 그리고 20 바이트인 경우에 대하여 나타낸 것으로, 기본 단위가 커지면 메모리 공간 낭비가 더욱 크게 발생할 수 있다. 그러나 스택 할당 및 반환의 숫자가 적어지므로 시간상의 오버헤드를 감소시킬 수 있게 된다.

다음의 알고리즘 2는 예측 버퍼에 기반한 동적 스택 관리 기법의 동작 방식을 나타내었다. 이 알고리즘에서, 각 함수 호출 시점에 먼저 검사하는 것이 스택의 잔량이다. 이전에 할당된 스택 공간이 호출될 함수가 수행되기에 충분한 공간을 갖고 있다면, 스택 할당이 일어나지 않게 된다. 반면에 스택 잔량이 부족하다면 새로이 스택을 할당하게 되고, 이후의 동작 방식은 알고리즘 1과 동일하다.

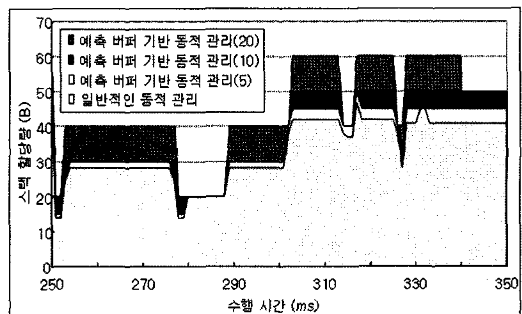


그림 4 예측 버퍼를 사용하는 기법과 일반 동적 스택 관리 기법의 스택 할당 비교

알고리즘 2. 예측 버퍼에 기반한 동적 스택 관리 방법
- 쓰레드가 sunny() 함수를 호출 할 때, 다음의 과정을 수행 -

```
1 IF spaceResidual < sunny().spaceRequired THEN
2 sunny().spaceRequired = big(spaceAlpha,sunny().
  spaceRequired)
3 sunny().addrStack = memAlloc(sunny().
  spaceRequired)
4 saveSP(sunny().addrStack)
5 changeSP(sunny().addrStack +sunny().
  spaceRequired -1)
6 push(sunny().arguments)
7 call(sunny())
8 loadSP(sunny().addrStack)
9 memFree(sunny().addrStack)
10 ELSE
11 push(sunny().arguments)
12 call(sunny())
13 END IF
```

4. 성능 평가

본 절에서는 멀티 쓰레디드 센서 운영체제를 위한 동적 쓰레드 스택 관리 기법의 성능 측정 결과를 보인다. 제안한 기법은 나노 Qplus 운영체제 상에서 구현되었다. 실험을 수행한 환경은 다음의 표 3에 보인다[21].

표 3 성능 평가를 위한 실험 환경

센서 플랫폼 구성	설명	비고
센서 운영체제	나노 Qplus 1.6.0e	ETRI 제작
센서 보드	Nano-24 센서 보드	Octacomm 제작
CPU	ATmega128L	8비트 프로세서
RAM	4KB	데이타+스택 영역
FLASH MEMORY	128KB	코드 영역
전원	2xAA 배터리	3.0 볼트

위의 표 3에서, Nano-24 센서 보드는 버클리 대학에서 설계한 MICAZ 센서 노드와 그 구성이 유사하다. 본 논문의 성능 평가에서는 Nano-24 보드를 사용하였고 제안한 기법의 구현을 위하여 일부의 커널 소스 코드를 수정 및 추가하였다.

동적 쓰레드 스택 관리 기법의 성능은 시간과 공간의 두 가지 측면에서 평가되어야 한다. 시간의 측면은 곧 수행 시간을 의미하고, 공간의 측면은 스택 메모리 할당량을 의미한다. 본 논문의 실험에서는, 이러한 시간 과 공간의 성능을 평가하기 위하여 제안한 두 가지 방식의 동적 스택 관리 기법과 기존의 정적 관리 기법에 대하여 스택 할당량과 전체 수행 시간을 비교하였다.

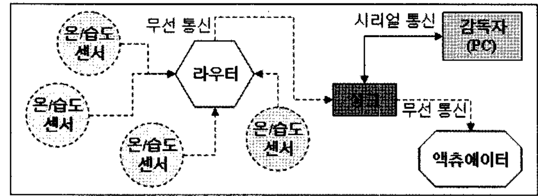


그림 5 "pthread_net4" 응용 예제의 구성 및 구조

그림 5는 성능 평가에서 사용한 "pthread_net4" 센서 네트워크 응용 프로그램의 구성을 나타낸 것이다. 총 네 가지 종류의 센서 노드들로 구성되고, 각각의 설명은 아래에 보인다.

- 온/습도 센서 노드: 주기적으로 온도 및 습도를 센싱하여, 그 결과를 라우터 노드에 전달하는 프로그램으로, 3개의 쓰레드를 사용한다.
- 라우터 노드: 센서 노드들로부터 받은 패킷을 싱크 노드로 전송하는 노드로, 2개의 쓰레드를 사용한다.
- 싱크 노드: 라우터 노드로부터 받은 패킷의 정보를 감독자의 PC에 시리얼 통신으로 출력하며, 온/습도의 결과를 분석하고, 액츄에이터 노드에 명령을 내린다. 총 2개의 쓰레드를 사용한다.
- 액츄에이터 노드: 싱크 노드로부터 받은 명령에 따라서 릴레이 회로를 여닫는 일을 수행한다. 2개의 쓰레드를 사용한다.

다음의 그림 6은 동적 스택 관리 기법을 사용하였을 때의 실행 중의 스택의 변화를 각 함수별로 나타낸 것이다. 이 네 가지의 결과 그래프는 각각 위의 네 가지 노드들에 해당되는 결과이다. 정적 스택 관리 기법을 사용하였을 경우에는 각 쓰레드에 고정적으로 128 바이트의 스택이 할당되므로 그래프로 나타내지 않았음을 밝힌다. 이 결과를 바탕으로, 동적 스택 관리 기법이 전체 메모리 공간을 정적 기법에 비하여 상당히 효율적으로 사용함을 확인할 수 있다. 또한, 함수의 실제 스택 사용량에 따른 동적 스택 할당이 잘 일어나는 것을 확인할 수 있다.

다음으로, 각 노드의 1주기 동안의 수행 시간을 기존의 정적 스택 관리 기법을 사용하였을 때와, 동적 스택 관리 기법을 사용하였을 때에 측정한 결과를 다음의 표 4에 보인다. 이 결과를 바탕으로, 전체 수행 시간 역시 그리 나빠지지 않음을 알 수 있다.

마지막으로, 3.3절에서 제안하였던 예측 버퍼에 기반한 동적 스택 관리 기법을 포함한 시간-공간 프로덕트를 다음의 그림 7에 보인다. 이 결과를 바탕으로, 시간상의 오버헤드와 공간의 효율성을 모두 고려하였을 때, 각 기법들의 성능을 한눈에 확인할 수 있게 된다.

위의 결과에서, 정적 스택 관리 기법에 비하여 동적

(a) 온/습도 센서 노드

(b) 라우터 노드

(c) 싱크 노드

(d) 액츄에이터 노드

그림 6 프로그램 실행 중의 스택 사용량의 변화

표 4 1주기 수행 시간의 비교

노드 구성	정적 스택 관리	동적 스택 관리
온/습도 센서	572 ms	595 ms
라우터	579 ms	603 ms
싱크	553 ms	597 ms
액츄에이터	471 ms	516 ms

그림 7 동적 스택 관리 기법의 시간-공간 프로덕트

관리 기법들은 모두 더 나은 성능을 보인다. 또한, 예측 버퍼에 기반한 동적 관리 기법은 할당의 기본 단위에 따라서 성능의 차이를 보이고 있음을 확인할 수 있다. 이 중에서, 액츄에이터와 라우터의 경우에는 5바이트의 기본 단위를 사용하는 예측 버퍼 기반 기법이 일반적인 동적 관리 기법보다 더 나은 성능을 보이고 있다. 이 예측 버퍼 기반 기법에서 사용되는 할당의 기본 단위와 관련한 연구는 향후에 지속적으로 연구될 것이다.

실험결과를 바탕으로, 공간 제약하의 멀티 쓰레디드 센서 운영체제를 위하여 제안한 동적 스택 할당 기법이 전체 공간을 효율적으로 사용함을 보였고, 수행 시간상의 오버헤드도 역시 그리 크지 않음을 확인하였다. 따라서, 무선 센서 네트워크용 운영체제들에서 이 기법을 사용하면, 멀티 쓰레디드 설계 방식의 단점인 메모리 사용량의 문제를 상당히 완화할 수 있을 것이다.

5. 결 론

무선 센서 네트워크는 수백 혹은 수천 개의 무선 센서 노드들로 이루어지고, 센서 노드의 플랫폼은 비용 효율성 때문에 매우 제한적인 메모리 공간을 지니며, 제한적인 배터리로 동작한다. 따라서 이것들을 동작시키는 센서 운영체제는 공간 제약성을 감내할 수 있어야 하고, 에너지 효율적으로 동작해야 센서 네트워크를 오랜 시간 동안 자원 효율적으로 동작시킬 수 있게 된다. 본 논문에서는 공간 제약하인 멀티 쓰레디드 센서 운영체제를 위한 동적 쓰레드 스택 관리 기법을 제안하였다. 제안한 기법은 각 쓰레드 함수의 스택 사용량을 측정하여, 측정 결과를 바탕으로 실행중에 동적으로 쓰레드 스택 영역을 할당 및 반환한다. 제안한 기법을 사용하면, 기

존의 정적인 스택 관리 기법들을 사용하는 것보다 스택 할당량을 상당히 감소시켜서, 전체 메모리 공간의 효율성을 증진시킬 수 있다. 본 논문의 비교 실험 결과를 통하여, 제안한 동적 스택 관리 기법을 사용하는 것이 기존의 기법보다 공간의 효율성을 상당히 개선시킬 수 있고, 또한 수행 시간의 오버헤드도 무시할만한 수준임을 보였다. 만약 제안한 기법이 실제 무선 센서 네트워크의 센서 운영체제들에서 사용된다면, 메모리 공간의 효율성을 효과적으로 증진시킬 수 있을 것이다.

본 논문의 향후 연구로는 다음의 것들이 존재한다. 먼저, 예측 버퍼에 기반한 동적 할당 기법에서, 할당의 기본단위에 대한 연구를 진행할 것이다. 이는 각 응용의 스택 사용량의 특성 정보를 사용하거나, 적응성 있는 할당 기본 단위의 결정 방법을 사용하게 하는 방식으로 동작하는 연구가 될 것이다. 또한, 기존의 스택 오버플로우 방지 기법들과 비교하여, MMU가 없는 시스템에서의 보다 효율적인 스택 오버플로우 방지 혹은 메모리 보호 기법 등의 연구를 진행할 것이다.

6. 소스코드 다운로드

제안한 메모리 할당 기법을 구현한 모든 소스코드와 실험에 사용한 코드는 다음의 URL에서 다운로드 할 수 있다.

http://ssrsec.snu.ac.kr/~shyi/kiss07stack.zip

참 고 문 헌

[1] C.A.R.Hoare, Monitors: An operating system structuring concept, Communications of the ACM 17, pp. 549-557, 1977.

[2] H.C. Lauer, R.M. Needham, On the duality of operating system structures, Second International Symposium on Operating Systems, IRIA, 1978.

[3] J.K. Ousterhout, Why threads are a bad idea (for most purposes), Presentation given at the 1996 Usenix Annual Technical Conference, 1996.

[4] A. Adya, J. Howell, M. Theimer, W.J. Bolosky, J.R. Douceur, Cooperative task management without manual stack management, Proceedings of the 2002 USENIX Annual Technical Conference, 2002.

[5] R. von Behren, J. Condit, E. Brewer, Why events are a bad idea (for high concurrency servers), HotOS IX: The 9th Workshop on Hot Topic in Operating Systems, pp. 19-24, 2003.

[6] A. Gustafsson, Threads without the pain, ACM Queue 3, pp. 34-41, 2005.

[7] P.Levis, S.Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, D. Culler, The emergence of networking abstractions and techniques in TinyOS, First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), 2004.

[8] C.C. Han, R. Kumar, R. Shea, E. Kohler, M.B. Srivastava, A dynamic operating system for sensor nodes, MobiSys, pp. 163-176, 2005.

[9] J. Mulder, S. Dulman, L. van Hoesel, P. Havinga, Peeros - system software for wireless sensor networks, Preprint, 2003.

[10] J. Yannakopoulos, A. Bilas, CORMOS: A communication-oriented runtime system for sensor networks, Proceedings of the 2nd IEEE European Workshop on Wireless Sensor Networks(EWSN), 2005.

[11] Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., Shucker, B., Gruenwald, C., Torgerson, A., Han, R.: MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms, ACMKluwer Mobile Networks and Applications(MONET) Journal, Special Issue on Wireless Sensor Networks, 2005.

[12] K. Lee, Y. Shin, H. Choi, S. Park, A design of sensor network system based on scalable and reconfigurable nano-os platform, IT-Soc International Conference, 2004.

[13] H. Kim, H. Cha, Towards a resilient operating system for wireless sensor networks, The 2006 USENIX Annual Technical Conference, 2006.

[14] A. Dunkels, B. Gronvall, T. Voigt, Contiki - a lightweight and flexible operating system for tiny networked sensors, First IEEE Workshop on Embedded Networked Sensors, 2004.

[15] Crossbow, http://www.xbow.com/.

[16] C. Tismer, Continuations and stackless python, Proceedings of the 8th International Python Conference, 2000.

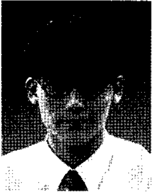
[17] P. Levis, D. Culler, Mate: a virtual machine for tiny networked sensors, International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 85-95, 2002.

[18] A. Dunkels, O. Schmidt, T. Voigt, Using protothreads for sensor node programming, Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks, 2005.

[19] A. Torgerson, Automatic thread stack management for resource-constrained sensor operating systems, BS-Thesis of University of Colorado, 2005.

[20] AVR-GCC, http://www.avrfreaks.net/AVRGCC/.

[21] Octacomm, http://www.octacomm.net/.



이 상 호

2003년 고려대학교 전기전자전파공학부 학사. 2003년~현재 서울대학교 컴퓨터공학부 박사과정(수료). 관심분야는 임베디드 시스템 및 임베디드 운영체제, 무선 센서 네트워크, 센서 운영체제, 결함 허용 컴퓨팅

조 유 근

1971년 서울대학교 건축공학과 학사. 1978년 미네소타대학교컴퓨터과학 박사. 1979년~현재 서울대학교 컴퓨터공학부 교수 1984년~1985년 미네소타대학교 교환 교수. 1993년~1995년 서울대학교 중앙교육연구전산원장. 1999년~2001년 서울대학교 공과대학 부학장. 2001년~2002년 한국정보과학회 회장. 관심분야는 운영체제, 시스템 보안, 암호학, 알고리즘 설계 및 분석, 무선 센서 네트워크.

홍 지 만

2003년 서울대학교 컴퓨터공학부 박사 졸업. 2004년~2007년 광운대학교 컴퓨터공학부 조교수. 2007년~현재 숭실대학교 컴퓨터학부 조교수. 관심분야는 임베디드 운영체제, 결함 허용 컴퓨팅, 무선 센서 네트워크