MMU가 없는 Thread기반 운영체제에서 스택 보호를 위한 메모리 관리 기법

이영림○, 김영필, 유혁
고려대학교
{yrlee○, ypkim, hxy}@os.korea.ac.kr

MMU-less Memory Management for Stack Protection in Thread-based Operating System

Youngrim Lee○, Youngpil Kim, Chuck Yoo
Dept. of Computer Science & Engineering, Korea University

요 약

현재 많은 센서 네트워크 운영체제에서는 메모리 제약 때문에 스레드 스택을 공유한다. 하지만 대부분의 대상 플랫폼에서는 MMU가 없어서 하드웨어적으로 스택 보호가 이루어지기 어렵다. 이러한 문제를 해결하기 위해 본 논문에서는 운영체제 바이너리 코드 안에 존재하는 스택 연산 명령어들을 스택 보호 기능을 가진 래퍼 함수호출로 바꾸어 주었다. 이 래퍼 함수는 스택의 오버플로우/언더플로우를 관리해 주고 오리지널 코드에 있던 명령어를 실행한 후 원래 실행 흐름으로 돌아가게 한다. 본 논문에서는 이러한 동작을 수행하는 Post-Compile Processing Tool의 구조와 세부 메커니즘을 제안한다. 이 툴은 직접 바이너리를 조작하므로 개발의 유연성을 살리고, 정적인 조작만 가하기 때문에 실행시간 오버헤드가 적다. 또한 임베디드 플랫폼 환경과 같이 하드웨어 자원의 제약이 있는 구조에 적합하다.

1. 서 론

센서 환경에서 하드웨어를 제어하고 그 위에서 동작하는 응용프로그램의 관리를 위해서 운영체제의 존재는 필수적이다. 현재 개발 되어 있는 센서 네트워크 운영체제는 프로그래밍 방식에 따라 이벤트 드리븐(event driven) 기반과 스레드(thread) 기반 방식으로 분류 할 수 있다. 이벤트 드리븐 기반의 센서 네트워크 운영체제에는 TinyOS[1], SOS[2]가 있고, 스레드 기반의 운영체제에는 MANTIS[3], PEEROS[4], ANTS[5], 나노 Qplus[6]가 있다. 스레드 기반 방식은 동시성(concurrency)를 제공해 줄 수 있는 반면 이벤트 드리븐 기반 방식은 이를 제공하지 못한다. 스레드 기반 방식은 컨텍스트 스위칭(context switching)과 같은 오버헤드를 가지는 반면 이벤트 드리븐 기반 방식은 이러한 오버헤드를 가지지 않는다. 현재로는 스레드 기반 방식이 각광받고 있다.[7]

멀티 스레드 시스템을 지원하는 센서 네트워크 운영체제에서는 다중 쓰레드들이 동작하는 스택(stack) 영역의 보호와 동적으로 메모리 블록을 할당 받거나 반납 하여 메모리 공간 사용의 효율을 높이는 메모리 관리가 필요하다.

현재도 많은 센서 네트워크 운영체제가 개발 중인데, 메모리 제약 때문에 스레드간의 스택을 공유한다. 그러나 대부분 대상 플랫폼(platform)에 하드웨어 MMU(Memory Management Unit)가 없어서 스택 보호(stack protection)가 이루어지기 어렵다. 이로 인해 센서 노드에서 동작하고 있는 프로그램들이 오동작하거나 시스템이 다운되는 문제가 발생한다.[8]

따라서 본 논문에서는 하드웨어 MMU가 없는 플랫폼 환경에서 동작하는 멀티 스레드 기반 운영체제에서 스택을 보호하기 위해 필요한 방법과 구조의 기본적인 설계를 제안한다. 제안하는 방법은 커널 개발의 유연성(flexibility) 위해 소스 코드를 전혀 수정하지 않으며, 툴을 이용하여 메모리 관리 기능을 추가 할 수 있다. 본 논문에서는 이 툴을 Post-Compile Processing Tool라고 명명한다.

본 논문의 구성은 다음과 같다. 2장에서는 제안한 아이디어의 전체 구조와 세부 메커니즘을 설명한다. 3장에서는 관련연구를 다룬다. 마지막 4장에서는 결론과 함께 향후 연구의 방향을 제시한다.

2. 설계

2.1 기본 설계

2.1.1 전체 구조

본 논문에서 스택 보호를 위해 설계한 Post-Compile Processing Tool의 구조는 다음 그림 1과 같다.

이 툴은 각각 바이너리 분석기(Binary Analyzer), 코드 생성기(Code Generator), 통합기(Integrator)로 구성 되며 이에 대한 설명은 다음과 같다.

● 바이너리 분석기(Binary Analyzer)

우리는 툴을 사용하는 정적인 방법으로 메모리 관리 기능을 추가 하였다. 이 툴에서 기존 컴파일이 된 바이너리(binary) 이미지에서 모든 스택 영역에 대한 접근을 파악하기 위해 필요하며, 바이너리 이미지 분석 작업을 통

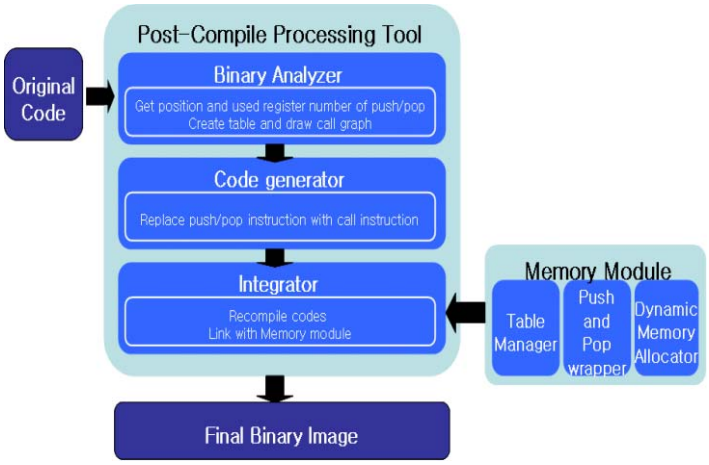


그림 1 Post-Compile Processing Tool 구조

해 이러한 접근에 사용되는 명령어를 메모리 보호가 가능한 래퍼(wrapper)함수의 호출로 재설정 한다.

이 바이너리 분석기의 기능은 다음과 같다.

첫째, 홀(hole)[9]을 찾기 위해 바이너리 이미지의 콜 그래프(call graph)를 만든다. 홀의 정의와 이를 찾는 이유에 대해서는 섹션2.2에서 자세히 다룬다.

둘째, 이 테이블은 래퍼 함수로 오기 전 원리 실행 하려 했던 push/pop 명령을 래퍼 함수에서 실행하기 위해 참조되는 테이블이다. 3개의 필드(field) 즉 명령어의 위치, 명령어의 종류(pop 인지 push인지 구분) 그리고 오퍼랜드(operand)인 레지스터를 저장하는 PUSH_POP 테이블을 생성한다. 자세한 메커니즘은 섹션 2.1.2에서 자세히 설명하겠다.

● 코드 생성기(Code generator)

코드 생성기는 바이너리 분석기에서 찾아낸 바이너리 이미지의 push/pop 명령어들을 래퍼 함수로 교체하기 위해 필요한 분기명령어를 생성하여 적용한다.

● 통합기(Integrator)

통합기는 코드 생성기에 의해 수정된 바이너리 이미지와 메모리 모듈(Memory Module)을 링킹(linking)하여 통합된 최종 커널 바이너리 이미지를 만든다.

다음으로 기본 설계의 핵심인 메모리 모듈(Memory module)에 대해 설명하겠다.

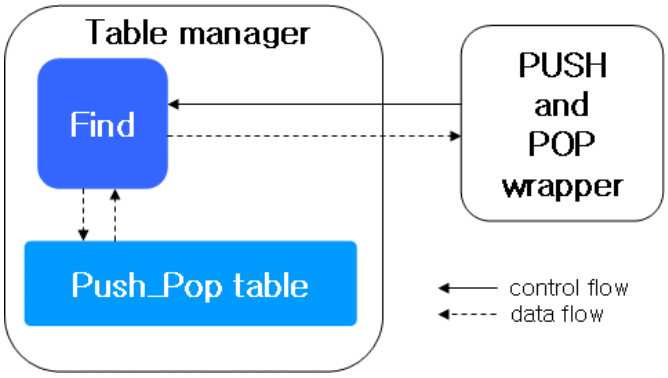


그림 2 Table Manager 구조

●테이블 관리자(Table manager)

테이블 관리자는 바이너리 분석기에 의해 생성된 PUSH_POP table을 관리하는 컴포넌트이다. 테이블 관리자의 전체적인 구조 및 동작은 그림 2와 같다.

PUSHandPOP 래퍼 컴포넌트에서 명령어의 offset을 넘겨주면, table manager의 find함수가 PUSH_POP table에서 해당 offset이 들어 있는 엔트리(entry)의 필드 값들을 PUSHandPOP 래퍼 컴포넌트에게 반환한다.

● PUSHandPOP 래퍼 컴포넌트

실제로 스택 보호를 수행하는 PUSHandPOP 래퍼 컴포넌트의 전체적인 동작 및 구조는 그림 3과 같다.

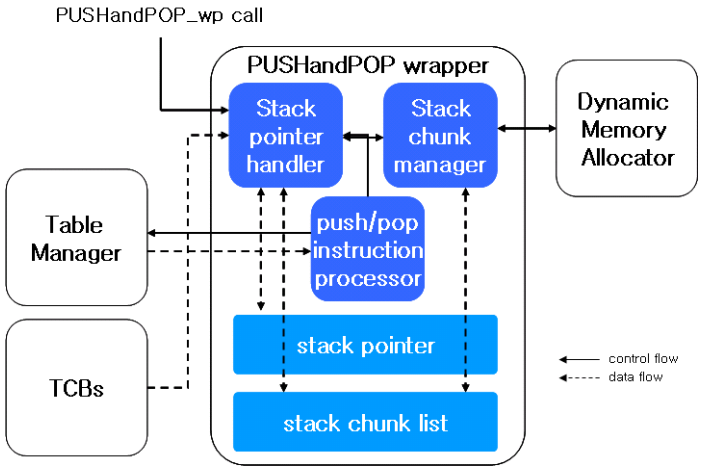
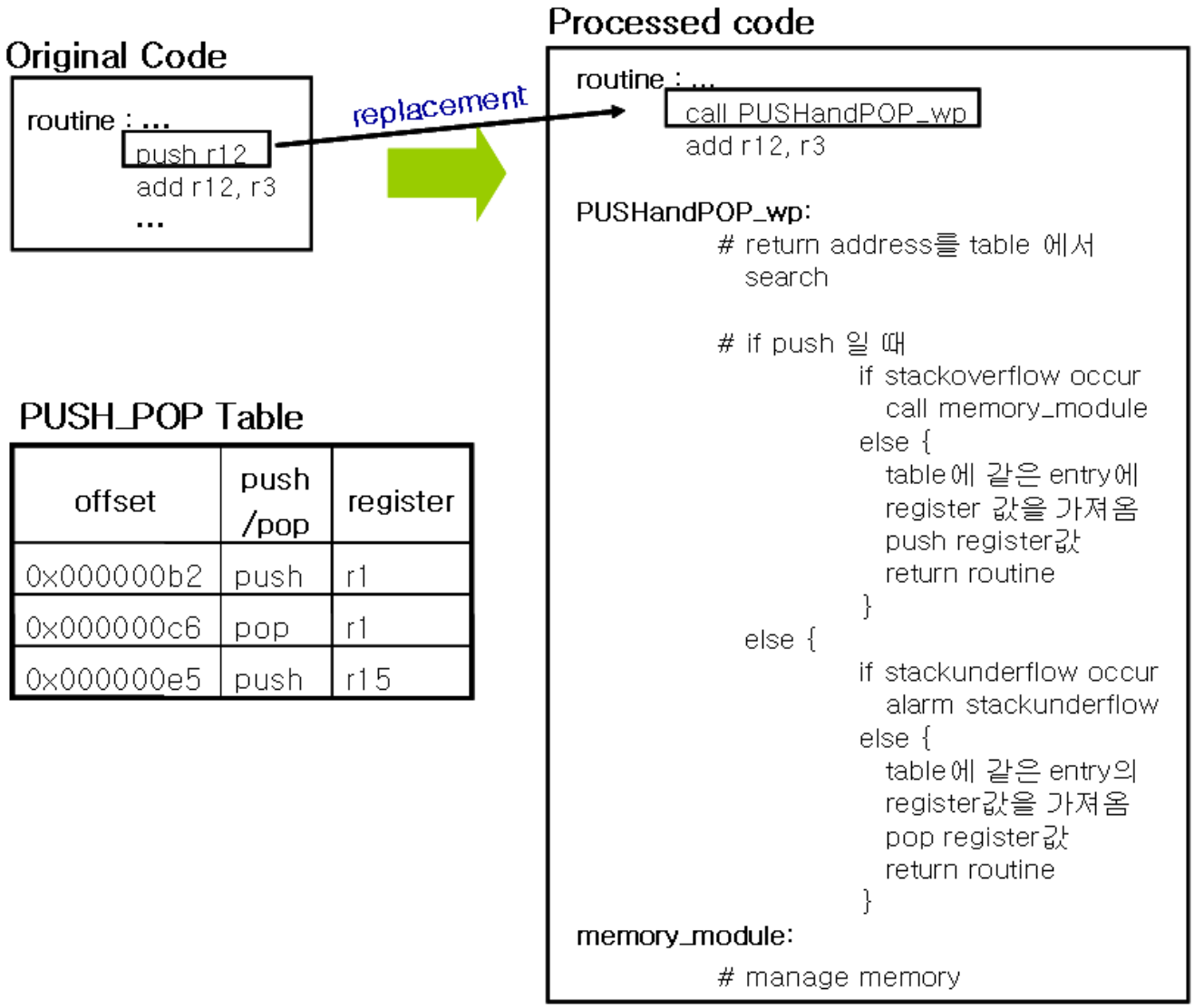


그림 3 PUSHandPOP wrapper 컴포넌트 구조

스택 보호의 핵심은 가용스택의 메모리 범위를 스택 포인터가 벗어나지 않도록 규제하는 것이다. 따라서 각 스레드의 스택 포인터를 관리하고 이를 처리하는 함수가 필요하며 이는 stack pointer handler이다. 또한 가용스택의 크기를 늘리면 스택에서 발생하는 오버플로우를 최소화 할 수 있다. 이를 위해 동적 메모리 할당이 요구되고, 이를 처리하는 함수가 stack chunk manager이다. Stack chunck manager는 동적 메모리 할당자에 의해서 할당받은 고정된 크기의 메모리 chunk를 linked list로 유지한다. 이러한 스택 보호 처리 후에 바이너리에에 있던 실제 push/pop 명령어를 수행할 함수가 push/pop instruction processor이다.

- 오버플로우시 PUSHandPOP 래퍼 동작

오버플로우는 스택으로 설정된 메모리 영역이 부족해서 발생한다. 본 논문에서는 동적 메모리 할당자로부터 메모리를 더 할당 받아 스택 크기를 늘려 오버플로우에 대처한다. 이때 스택 포인터를 할당받은 메모리를 가리키도록 조작하기 때문에 현재 동작하는 스레드를 알아야 한다. 스레드의 스택 포인터는 TCB(Thread Control Block)에 들어있다. PUSHandPOP 래퍼가 스택을 관리하는 자세한 연산과정은 다음과 같다. PUSHandPOP 래퍼가 호출되면 Stack pointer handler가 TCB로부터 현재 스레드 정보와 스택 포인터(stack pointer)를 가져온다. 오버플로우가 발생하면 stack pointer handler는 stack chunk manager에게 메모리 chunk 할당을 요청한다. stack chunk manager는 메모리 chunk 할당 요청을 받으면, Dynamic Allocator에게 메모리 할당요청을 한



PUSH_POP Table

offset	push /pop	register
0x000000b2	push	r1
0x000000c6	pop	r1
0x000000e5	push	r15

그림 4 세부 메커니즘

다. stack chunk manager는 Dynamic Allocator에게서 메모리 할당받으면 stack chunk list에 할당 받은 메모리 chunk를 넣는다. 이를 stack pointer handler에게 알린다. stack pointer handler는 스택 포인터가 할당받은 메모리 chunk를 가리키도록 한다. 스택 보호 관련 연산이 완료되었기 때문에, 이 후에는 정상적인 스택 관련 연산을 수행해 주어야 한다. push/pop instruction processor에서 오리지널 코드(original code)에 있었던 push/pop 명령을 실행하기 위해 테이블 관리자와 상호작용을 통해 명령어의 종류(push/pop인지 구분)와 그 오퍼랜드의 값을 받는다. push/pop instruction processor는 오퍼랜드를 가지고 오리지널 코드에 있던 명령어를 수행한다. 그리고 원래 실행 흐름(flow)으로 돌아간다.

2.1.2 세부 메커니즘

위의 전체 구조에서 세부적으로 어떤 메커니즘을 가지고 동작하고 있는지 살펴보자.(그림 4)

● 스택 오버플로우/언더플로우 처리

PUSHandPOP 래퍼 함수에서 복귀 주소를 PUSH_POP table에서 찾은 후, 일치하는 엔트리의 push/pop 필드로 push인지 pop인지 판별한다. push라면 스택 포인터를 가지고 스택 오버플로우가 발생했는지 아닌지를 판별한다. 스택 오버플로우가 발생한다면 동적으로 메모리를 할당해제 해주는 메모리 모듈을 호출한다. 스택 포인터가 할당 받은 메모리 chunk의 시작위치를 가리키게 한다. pop라면 스택 언더플로우가 발생했는지 검사하고 발생한 경우 이를 알리고, 해당 스레드를 종료하거나, 그 스레드를 다시 실행 한다.

● 오더/언더플로우 처리 후 원래 스택 연산의 수행

PUSH_POP table에서 push 혹은 pop을 할 대상인 레지스터를 가져온다. 그 값을 가지고 push/pop 연산을 수행 후에 원래 루틴(routine)으로 돌아간다.

2.2 실제 target 플랫폼에 적용

실제 사용되고 있는 플랫폼과 센서 네트워크 운영체제에 이 논문에서 제안한 스택 보호 방법을 적용해 보겠다.

본 논문에서는 한국전자통신 연구원(ETRI)에서 개발된 나노 Qplus의 업그레이드된 버전인 Nano OS와 옥타 Nano-24를 대상으로 하였다.

2.2.1 적용 시 발생한 문제점

첫 번째 문제점은 분기 명령어 길이 차이에 의한 명령어 밀림현상이다. 코드 생성기에서 push 명령어를 call 명령어로 바꾸면 push명령어 다음의 명령어가 corruption 되지 않게 하기 위해 이후 모든 명령어들을 뒤로 밀게 된다.

target 플랫폼은 2byte 정렬이다[10]. 하지만 여기서는 예외적으로 call 명령어의 크기는 4byte 이다. 그래서 우리가 실제로 2byte 크기의 push/pop명령어 자리에 push/pop 명령어 대신 call 명령을 쓰게 되면, 그 뒤 모든 명령어의 주소를 뒤로 밀어야 하고, 메모리 offset의 재조정을 해야 한다. 그 이유는 만약 밀리는 명령어 중 unconditional 혹은 conditional jump 혹은 call 명령어가 있다면, 이 명령어의 오퍼랜드인 주소 값이 원래 정해진 곳이 아니라 밀림 현상에 의해서 엉뚱한 곳의 값을 가지고 있는 것으로 되어버리기 때문이다.

이것을 해결하기 위해 call 명령어 대신 rcall 명령어를 사용한다. call 명령어는 4byte를 차지하지만, rcall 명령어는 2byte만 차지한다. 이렇게 함으로써 밀림현상은 방지 할 수 있지만 rcall 명령어가 도달할 수 있는 offset은 -2K <= offset <= 2K 이기 때문에 이를 벗어난 위치의 래퍼 컴포넌트에 바로 도달할 수 없다. 그래서 우리는 SPRING을 이용한다. SPRING은 rcall로 도달 가능한 영역에 위치하며 그 구조는 다음과 같다.

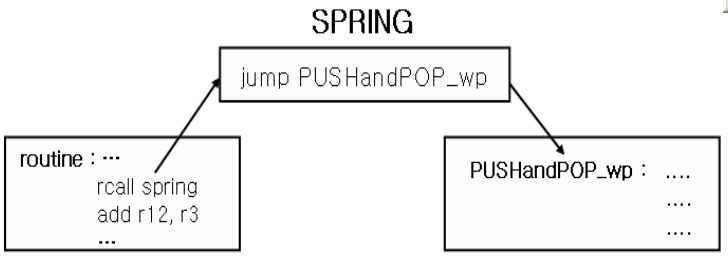


그림 5 SPRING

SPRING은 모든 메모리에 접근 가능한 분기 명령어로 되어있으며, rcall 명령어의 offset으로 도달 가능한 홀에 위치한다. 여기서 홀이란 실행 플로우(flow)에서 도달하지 않은 지역(unreachable region)이다.[9] 본 논문에서는 SPRING의 크기인 4byte 크기의 hole이 원래 바이너리 이미지에 반드시 존재한다고 가정한다. 이 홀은 Binary analyser에서 콜 그래프(call graph)를 그려서 찾아 낼 수 있다. 본 논문의 대상인 Nano OS에서는 interrupt vector table을 홀로 이용하였다.(그림 6)

두 번째 문제점은 push/pop 명령어 이 외에도 스택을 조작하는 명령어가 있어서 발생하는 오버플로우/언더플로우의 처리이다. 일례로 rcall 명령어를 실행하면 복귀 주소 역시 스택에 들어가며, 이때 스택 오버플로우가 발생할 수도 있다.

이를 해결하는 방법은 스택에 저장되는 복귀 주소를 고려하여 현재 오버플로우/언더플로우의 체크 바운더리(boundary)를 낮추거나 높이는 것이 될 수 있다.

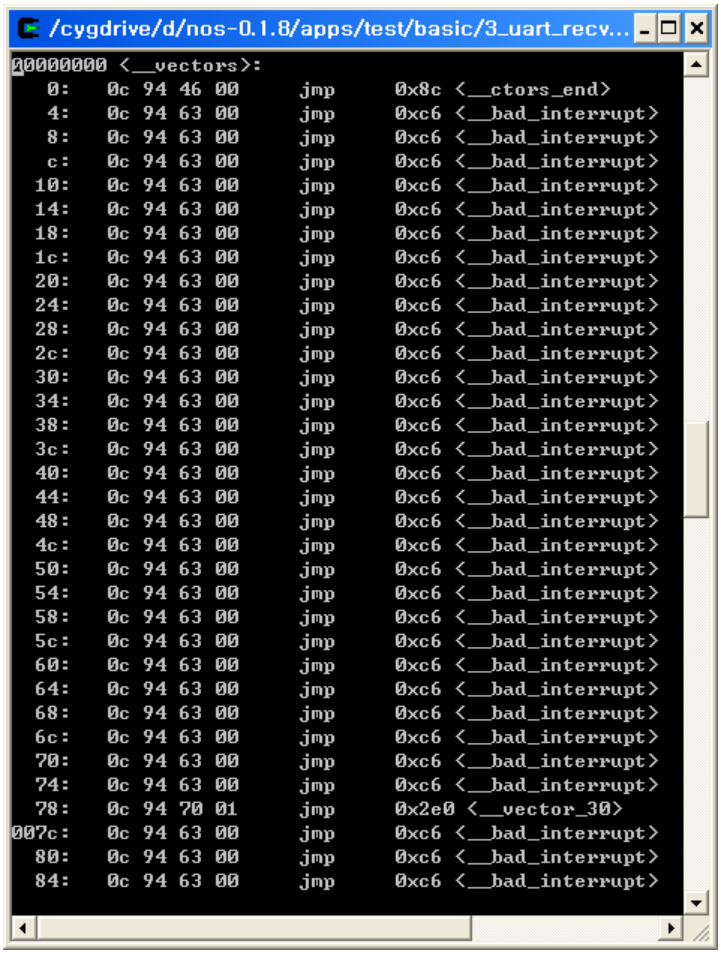


그림 6 Nano OS의 interrupt vector table

3. 관련 연구

하드웨어 MMU가 없는 플랫폼에서 MMU를 지원하는 방법에는 여러 가지 접근 방법이 있다.

첫 번째, 하드웨어 MMU를 가진 시스템을 모방하는 접근 방식이 있다. 이 접근 방식은 응용프로그램에는 수정을 가하지 않지만. 모든 메모리 접근 시 가상 주소를 물리 주소로 바꿔주는 함수를 호출해야 하는 단점이 있다. Softvm[11]은 이 접근 방식을 사용했다. Softvm의 기본 아이디어는 virtually indexed, virtually tagged cache 계층 구조에서 캐쉬 미스(cache miss)가 발생하면, 캐쉬 미스 핸들러(cache miss handler)가 주소를 변환주며, 물리 주소의 데이터를 가져오는 것이다. Softvm은 대상 하드웨어가 반드시 virtually indexed, virtually tagged cache 계층 구조를 가지고 있어야 한다는 제약사항이 있다. 또한 캐쉬 미스를 소프트웨어 적으로 관리할 수 있는 메커니즘이 필요하다.

두 번째, [12]에서는 컴파일러를 이용하여 가상 메모리를 관리한다. vm_assembler를 이용하여 어셈블리 코드에 존재하는 메모리와 관련된 명령어를 가상 주소 변환 함수(가상 주소를 물리 주소로 바꿔주는 함수)를 호출하는 명령어로 바꿔준다.

본 연구는 두 번째 접근 방식과 유사하나 가상 메모리 시스템의 전체 기능을 구현하지 않고, 현실적으로 보호가 필요한 스택에 초점을 두었다는 점에서 차이가 있다.

4. 결론 및 향후 연구

본 논문은 MMU가 없는 플랫폼에서 스택 보호를 하기 위하여 메모리 관리 기능을 추가할 수 있게 해주는 Post-Compile Processing Tool의 기본 구조와 동작 메커니즘을 제안하였다. 이 툴은 직접 바이너리를 조작하므로 개발의 유연성을 살릴 수 있고, 정적인 조작만 가하므로 실행시간 오버헤드가 적다. 이는 임베디드 환경과 같이 잦은 빌드를 요하고, 플랫폼의 하드웨어 자원의 제약이 있는 구조에 적합하다.

추후연구로는 제안한 방법을 좀 더 최적화 할 예정이다. 본 논문에서는 push/pop과 같이 명시적인 stack 조작 명령어만 처리 했는데 좀 더 발전시켜서 다른 스택 조작 명령어들을 처리할 수 있도록 하겠다. 또한 동적 메모리 관리 방법과 본 논문에서 제안한 툴의 구현을 진행할 예정이다.

참고 문헌

[1] LEVIS, P., MADDEN, S., GAY, D., POLASTRE, J., SZEWCZYK, R., WOO, A., BREWER, E., AND CULLER, D. The emergence of networking abstractions and techniques in tinyos. In Proceedings of the First Symposium on Networked Systems Design and Implementation (2004), USENIX Association, pp. 1-14.

[2] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, A dynamic operating system for sensor nodes. In MobiSYS '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services, 2005.

[3] H. Abrach , S. Bhatti , J. Carlson , H. Dai , J. Rose , A. Sheth , B. Shucker , J. Deng , R. Han, MANTIS: system support for multimodAl NeTworks of in-situ sensors, Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications, September 19-19, 2003, San Diego, CA, USA

[4] J. Mulder, S. Dulman, L. van Hoesel, and P. Havinga. PEEROS --- System Software for Wireless Sensor Networks. Preprint, August 2003

[5] KIM Daeyoung; TOMAS SANCHEZ LOPEZ; YOO Seongeun; SUNG Jongwoo; KIM Jaeeon, KIM Youngsoo; DOH Yoonmee, "ANTS : An evolvable network of tiny sensors", Lecture notes in computer science (Lect. notes comput. sci.) ISSN 0302-9743

[6] Seungmin Park, Jin Won Kim, Kwangyong Lee, Kee-Young Shin, Daeyoung Kim, "Embedded Sensor Networked Operating System", Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06) pp. 117-124

[7] Rob von Behren, Jeremy Condit and Eric Brewer, "Why Event Are A Bad Idea", the 9th Workshop on Hot Topics in Operating System, Lihue, Hawaii, USA, May 18-21, 2003

[8] John Regehr, Nathan Cooprider, Will Archer, Eric Eide,"Memory Safety and Untrusted Extensions for TinyOS", Technical report, UUCS-06-007, School of Computing University of Utah Salt Lake City, UT84112 USA

[9] Tarek Alameldin and Tarek Sobh, "On the evaluation of reachable workspace for redundant manipulators", Proceedings of the 3rd international conference on Industrial and engineering applications of artificial intelligence and expert systems - Volume 2 IEA/AIE '90

[10] atmega128 datasheet, available : http://www.atmel.com/

[11] Bruce Jacob, "Uniprocessor Virtual Memory without TLBs", IEEE Transaction on Computers, vol. 50, No. 5, May 2001

[12] Siddharth Choudhuri, "Software Virtual Memory Management for MMU-less Embedded Systems", Technical Report CECS-05-16, 2005.