논문 2018-13-22

Zephyr 커널에서 커널 공간과 사용자 공간의 분리 구현

(Separation of Kernel Space and User Space in Zephyr Kernel)

김 은 영, 신 동 하*
(Eunyoung Kim, Dongha Shin)

Abstract : The operating system for IoT should have a small memory footprint and provide low power state, real-time, multitasking, various network protocols, and security. Although the Zephyr kernel, an operating system for IoT, released by the Linux Foundation in February 2016, has these features but errors generated by the user code can generate fatal problems in the system because the Zephyr kernel adopts a single-space method that both the user code and kernel code execute in the same space. In this research, we propose a space separation method, which separates kernel space and user space, to solve this problem. The space separation that we propose consists of three modifications in Zephyr kernel. The first is the code separation that kernel code and user code execute in each space while using different stacks. The second is the kernel space protection that generates an exception by using the MPU (Memory Protection Unit) when the user code accesses the kernel space. The third is the SVC based system call that executes the system call using the SVC instruction that generates the exception. In this research, we implemented the space separation in Zephyr v1.8.0 and evaluated safety through abnormal execution of the user code. As the result, the kernel was not crashed by the errors generated by the user code and was normally executed.

Keywords : Space separation, Kernel protection, System call, Cortex-M4, Zephyr

Ⅰ. 서 론

최근 사물인터넷 (IoT) 기술이 다양한 분야에서 활용되면서 사물인터넷용 운영체제 [1, 2]의 중요성이 증가하고 있다. 사물인터넷용 운영체제는 기존 운영체제와 달리 적은 메모리 사용, 저 전력 기능, 실시간성, 멀티태스킹, 다양한 네트워크 프로토콜, 안전성 등을 제공해야 한다. 2016년 2월 리눅스 재단에서는 이러한 기능들을 제공하는 사물인터넷용 운영체제인 Zephyr 커널 [3]을 발표하였다. Zephyr 커널은 자원이 한정적인 기기를 위해 설계된 small-footprint 실시간 커널이며 SoC 칩에 내장된 저 전력 기능을 최대한 활용할 수 있도록 전력 관리 서브시스템을 제공한다. 또한 Zephyr 커널은 실시간 커널로 커널 서비스를 항상 일정한 시간 내에 완료하고, 라운드 로빈 스케줄링을 사용하는 멀티쓰레딩을 제공하고, IEEE 802.15.4, IPv4/IPv6, TCP/UDP, HTTP, CoAP 등의 다양한 네트워크 프로토콜을 제공한다.

Zephyr 커널은 커널 코드와 사용자 코드를 함께 링크하여 하나의 이미지를 생성하며 두 코드는 단일 공간에서 수행된다 [3]. 단일 공간 방식은 커널 코드를 간소화하고 시스템 호출을 함수 호출 방식으로 수행할 수 있도록 하지만 사용자 코드에서 발생한 오류가 같은 공간에서 수행중인 커널에 영향을 주어 시스템 전체에 치명적인 문제를 일으킬 수 있다. 예를 들어 사용자 코드에서 발생한 잘못된 메모리 접근은 하드웨어 오류, 커널 데이터 손상, 의도적이지 않은 코드의 실행 등의 문제를 발생시킨다 [4, 5]. 최근 Zephyr 커널은 커널의 안전성을 향상시키기 위하여 향후 버전에 메모리 분리, 스택 및 쓰레드 분리 등을 구현할 예정이라는 계획을 발표하였으나 아직 자세한 정보는 공개되지 않았다 [3].

*Corresponding Author (dshin@smu.ac.kr)
Received: Feb. 2 2018, Revised: Apr. 19 2018, Accepted: June 29 2018.
E. Kim, D. Shin: Sangmyung University
※ 본 연구는 2018년도 상명대학교 교내연구비를 지원받아 수행하였음.

ⓒ IEMEK J. Embed. Sys. Appl. 2018 Aug. 13(4) 187-194
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2018.13.4.187

본 연구에서는 이러한 문제를 해결하기 위하여 공간 분리 방식 [6]을 제안한다. 본 연구에서 제안하는 공간 분리는 커널 공간과 사용자 공간을 분리시켜 사용자 코드에서 발생한 오류가 커널 공간에 영향을 줄 수 없도록 하며 코드 분리, 커널 공간 보호, SVC 방식의 시스템 호출로 구성된다. 여기서 코드 분리는 커널 코드와 사용자 코드가 분리되어 각자의 공간에서 수행되면서 서로 다른 스택을 사용하는 것이고, 커널 공간 보호는 MPU (Memory Protection Unit)를 사용하여 사용자 코드가 커널 공간에 접근하면 예외를 발생시키는 것이고, SVC 방식의 시스템 호출은 예외를 발생시키는 SVC 명령어 [7-9]를 사용하여 시스템 호출을 수행하는 것이다.

본 연구에서는 공간 분리를 Zephyr v1.8.0에 구현하였으며 Cortex-M4 프로세서 [8, 9]가 탑재된 NXP사의 FRDM-K64F 보드 [10-12]에서 구현하여 안전성을 시험하였다. 시험 결과 커널에 문제를 일으킬 수 있는 사용자 코드의 수행이 기존 Zephyr 커널에서는 커널에 영향을 주어 커널이 복구되지 않는 반면에 본 연구에서 제안하는 공간 분리가 구현된 Zephyr 커널에서는 커널에 영향을 주지 않아 커널이 정상적으로 수행되었다. 공간 분리 방식은 코드의 복잡성과 효율성 면에서 단점이 있지만 안전성 시험의 결과처럼 커널의 안전성을 향상시키기 때문에 사용자는 공간 분리 방식을 사용할지 여부를 자신의 개발 환경과 응용 프로그램의 특성을 고려하여 선택할 수 있다.

본 논문의 2장에서는 사전 연구로서 ARMv7-M 구조의 동작 모드, 스택 포인터, FRDM-K64F 보드의 마이크로컨트롤러에 내장된 MPU 그리고 ARMv7-M 구조의 SVC 예외의 처리 과정에 대하여 설명한다. 3장, 4장 및 5장에서는 본 연구에서 제안하는 공간 분리를 구성하는 코드 분리, 커널 공간 보호, SVC 방식의 시스템 호출을 구현하는 방법에 대하여 설명한다. 6장에서는 안전성 시험 결과와 에너지 소모량에 대하여 설명하며 마지막으로 7장에서 본 논문의 결론을 기술한다.

II. 사전연구

이 장에서는 Zephyr 커널에 공간 분리를 구현하는데 필요한 사전 연구에 대하여 설명한다. 먼저 코드 분리, 커널 공간 보호, SVC 방식의 시스템 호출 구현의 기반 기술인 ARMv7-M 구조의 동작 모드(operation mode)와 스택 포인터 [7]에 대하여 설명하고 커널 공간 보호를 구현하는데 필요한 FRDM-K64F 보드의 마이크로컨트롤러 (MCU)에 내장된 MPU [11]에 대하여 설명한다. 마지막으로 SVC 방식의 시스템 호출을 구현하는데 필요한 ARMv7-M 구조의 SVC 예외 처리 과정 [7, 9]을 설명한다.

1. ARMv7-M 구조의 동작 모드와 스택 포인터

ARMv7-M 구조의 동작 모드와 스택 포인터는 공간 분리를 구성하는 코드 분리, 커널 공간 보호, SVC 방식의 시스템 호출 구현을 위한 기반 기술이다. 동작 모드와 스택 포인터는 코드 분리가 구현되면 커널 코드와 사용자 코드가 사용하는 스택이 다르다는 것을 이해하는데 필요하고, 커널 공간 보호를 구현할 때 커널 공간의 접근 권한을 설정하는데 필요하고, SVC 방식의 시스템 호출을 구현할 때 SVC 예외 핸들러를 구현하는데 필요하다.

ARMv7-M 구조의 동작 모드에는 핸들러 모드 (handler mode)와 쓰레드 모드 (thread mode)가 있다. 핸들러 모드는 항상 실행 레벨 (execution level)이 특권 (privileged)이고 스택 포인터로 MSP (Main Stack Pointer)를 사용한다. 쓰레드 모드는 실행 레벨을 특권 혹은 비특권 (unprivileged)으로 설정할 수 있고, 사용할 스택 포인터를 MSP 혹은 PSP (Process Stack Pointer) 중에서 설정할 수 있다.

2. FRDM-K64F 보드의 MCU에 내장된 MPU

FRDM-K64F 보드의 MCU는 Cortex-M4 프로세서에 내장된 MPU를 사용하지 않고 자체 MPU를 사용한다 [11]. 본 연구에서는 공간 분리가 구현된 Zephyr 커널을 FRDM-K64F 보드에서 수행시키기 때문에 공간 분리를 구성하는 커널 공간 보호를 구현하기 위해 MCU에 내장된 MPU를 사용했다. 이 MPU는 4개의 레지스터로 구성된 12개의 영역 디스크립터 (region descriptor)를 사용하여 메모리를 최대 12개의 영역으로 분할할 수 있다. 영역 디스크립터는 MPU_RGDn_WORD0-3 레지스터로 구성되며 각각 메모리 영역 시작 주소, 끝 주소, 접근 권한, 영역 디스크립터의 유효 비트 등을 저장한다. 하나의 메모리 영역에 대하여 접근 권한을 정의할 때 사용자는 실행 레벨이 특권 레벨일 때의 접근 권한과 비특권 레벨일 때의 접근 권한을 다르게 정의할 수 있다. 만약 사용자가 접근 권한이 없는 메

모리 영역에 접근하면 MPU는 Cortex-M4 프로세서의 fault 중에서 BusFault를 발생시킨다.

3. ARMv7-M 구조의 SVC 예외 처리 과정

본 연구에서는 SVC 방식의 시스템 호출을 구현하기 위해 먼저 SVC 예외의 처리 과정을 분석하였다. ARMv7-M 구조에서 SVC 명령어가 수행되면 SVC 예외가 발생되며 이 예외를 처리하기 위해 하드웨어가 다음을 수행한다. 먼저 현재 사용 중인 스택 포인터가 가리키는 스택에 xPSR, 반환 주소, LR, R12, R3-R0 레지스터의 현재 값을 저장하고, LR에 현재 동작 모드와 사용 중인 스택 포인터의 정보가 담긴 EXC_RETURN 값을 저장한다. 그 다음에 동작 모드를 핸들러 모드로 변경하고, 스택 포인터를 MSP로 변경하고, 벡터 테이블에 저장된 SVC 예외 핸들러의 주소를 PC에 저장하여 SVC 예외 핸들러를 수행시킨다. 나중에 핸들러가 반환되면 하드웨어는 PC에 LR의 값을 저장하여 동작 모드와 스택 포인터를 복구하고, 복구된 스택 포인터가 가리키는 스택에서 레지스터들의 값을 복구한다.

Ⅲ. 코드 분리

코드 분리는 커널 코드와 사용자 코드를 분리하여 각 코드가 각자의 공간에서 수행되면서 서로 다른 스택을 사용하도록 하는 것이다. 기존의 Zephyr 커널은 컴파일 시 커널 코드와 사용자 코드를 단일 공간에 모두 링크하며 커널과 사용자가 같은 스택을 사용한다. 본 연구에서는 코드 분리를 구현하기 위하여 컴파일 시 커널 코드와 사용자 코드가 각자의 공간으로 링크되도록 수정하였으며 커널과 사용자가 사용하는 스택을 분리하였다. 다음은 코드 분리를 구현하기 위한 컴파일 분리와 스택 분리에 대한 설명이다.

1. 컴파일 분리

컴파일 분리는 커널 코드와 사용자 코드를 각자의 공간으로 링크하는 것이다. 컴파일 분리를 구현하려면 먼저 기존의 단일 공간이 커널 공간과 사용자 공간으로 분리되어야 한다. 커널 공간은 커널 코드가 저장되는 곳과 커널 데이터가 저장되는 곳으로 구성되며 FRDM-K64F 보드의 MCU의 메모리 맵 [11]에서 FLASH 영역과 SRAM_U 영역이 커널 공간에 해당된다. 사용자 공간은 사용자 코드와 사용자 데이터가 저장되는 곳으로 SRAM_L 영역이 사용자 공간에 해당된다. 그 다음에 커널 코드와 사용자 코드가 각자의 공간으로 링크되도록 Zephyr 커널의 컴파일 과정을 다음과 같이 변경하였다. 컴파일 시 사용자 이미지와 커널 이미지를 각각 생성하고, 두 이미지를 합쳐 하나의 이미지 파일을 만들도록 하였다. 생성된 이미지 파일은 메모리의 FLASH 영역에 저장되기 때문에 나중에 커널 초기화 시 사용자 이미지 파일의 내용이 사용자 공간으로 자동으로 복사되도록 하였다.

2. 스택 분리

기존 Zephyr 커널에서는 함수 호출 방식으로 시스템 호출을 수행하기 때문에 커널이 시스템 호출을 수행하는 동안에는 PSP가 가리키는 사용자 쓰레드의 스택을 사용한다. 코드 분리가 구현되면 사용자 코드에서 함수 호출 방식으로 시스템 호출을 수행할 수 없기 때문에 시스템 호출을 수행하는 동안 커널이 사용자 쓰레드의 스택을 사용할 수 없다. 본 연구에서는 커널이 시스템 호출을 수행할 때에도 커널 스택을 사용하도록 하여 커널과 사용자가 사용하는 스택을 완전히 분리시켰다. 이를 위해 본 연구에서는 SVC 방식의 시스템 호출을 구현하였으며 이에 대하여 5장에서 자세히 설명한다.

Ⅳ. 커널 공간 보호

커널 공간 보호는 MPU를 사용하여 사용자 코드가 커널 공간에 접근하면 예외를 발생시키는 것이다. 기존의 Zephyr 커널은 사용자 코드가 커널 코드와 같은 공간에서 수행되기 때문에 사용자 코드에서 커널 코드나 데이터에 접근할 수 있다. 본 연구에서는 커널 공간 보호를 구현하기 위하여 커널 코드의 실행 레벨을 특권으로 설정하고, 사용자 코드의 실행 레벨을 비특권으로 설정하고, MPU를 사용하여 커널 공간의 접근 권한을 새로 정의하여 사용자 코드가 접근할 수 없도록 하였다. 다음은 커널 공간 보호를 구현하기 위한 MPU 설정과 BusFault 예외 핸들러에 대한 설명이다.

1. MPU 설정

사전 연구의 2절에서 설명한 바와 같이 MPU를 사용하여 메모리 영역의 접근 권한을 설정할 때 실행 레벨에 따라 접근 권한을 설정할 수 있다. Zephyr 커널에서 커널 코드는 핸들러 모드에서 수

```
1 SVC_FUNC(P1, P2, P3, P4)
2 {
3    svc syscall_number;
4    return R0;
5 }
```

그림 1. SVC 호출 함수의 의사코드
Fig. 1 Pseudo code for SVC call function

행되고 사용자 코드는 쓰레드 모드에서 수행된다. 기존에는 쓰레드 모드의 실행 레벨을 특권으로 설정하는데 본 연구에서는 쓰레드 모드의 실행 레벨을 비특권으로 설정하였다. 그 다음에 커널 공간의 접근 권한을 실행 레벨이 특권일 때에는 읽기(load), 쓰기 (store), 명령어 페치 (instruction fetch)가 모두 가능하도록 설정하고, 비특권일 때에는 접근이 불가능 하도록 설정하였다. 이를 위해 커널 공간의 접근 권한을 정의하는 코드를 작성하여 커널 공간을 정의하는 메모리 영역 디스크립터의 MPU_RGDn_WORD2 레지스터에 저장하였다. 또한 사용자 공간에 해당하는 메모리 영역의 접근 권한을 정의하는 코드도 작성하여 사용자 공간에서 실행 레벨에 상관없이 읽기, 쓰기, 명령어 페치가 모두 가능하도록 설정하였다.

기존에는 커널 메인 쓰레드가 사용자 프로그램의 main 함수를 호출하기 때문에 main 함수가 수행되는 동안 커널 메인 쓰레드의 스택이 사용된다. 본 연구에서는 사용자 코드의 main 함수가 실행 함수이고 사용자 공간에 구현된 스택을 사용하는 사용자 메인 쓰레드를 커널이 생성하도록 수정하여 사용자 코드에서 사용자 쓰레드만 수행되도록 하였다.

2. BusFault 예외 핸들러

사용자 코드가 커널 공간에 접근하면 사전 연구의 2절에서 설명한 바와 같이 MPU에 의하여 BusFault [7, 9]가 발생한다. BusFault 예외 핸들러에서는 현재 수행중인 쓰레드의 ID, fault를 발생시킨 명령어의 주소, fault의 원인을 화면에 출력한다. 그 다음에 fault가 발생하기 전의 상태에 따라 이후에 커널이 수행할 동작을 다음과 같이 결정한다. fault가 발생하기 전에 ISR 혹은 커널 쓰레드가 수행 중 이었다면 커널을 idle 상태로 만들어 시스템을 중단시킨다. 사용자 쓰레드가 수행 중 이었다면 현재 수행중인 쓰레드를 강제로 종료시키고 다른 쓰레드를 수행시킨다. 본 연구에서는 사용자 코드에서 사용자 쓰레드만 수행되도록 수정하였기 때문에 사용자 코드가 실수 혹은 의도적으로 커널 공간에 접근하여 BusFault가 발생하면 현재 수행중인 쓰레드만 종료되고 커널은 정상적으로 수행된다.

Ⅴ. SVC 방식의 시스템 호출

SVC 방식의 시스템 호출은 예외를 발생시키는 SVC 명령어를 사용하여 시스템 호출을 수행하는 것이다. 기존의 Zephyr 커널은 함수 호출 방식으로 시스템 호출을 수행하지만 코드 분리와 커널 공간 보호가 구현되면 사용자 코드에서 함수 호출 방식으로 시스템 호출을 수행할 수 없다. 본 연구에서는 사용자 코드에서 시스템 호출을 수행할 수 있도록 SVC 방식의 시스템 호출을 구현하였다. 다음은 SVC 방식의 시스템 호출을 구현하기 위한 SVC 호출 함수, SVC 예외 핸들러에 대한 설명이다.

1. SVC 호출 함수

시스템 호출을 SVC 방식으로 수행하기 위해선 사용자 코드에서 시스템 호출을 호출할 때마다 SVC 예외가 발생해야 한다. 이를 위해 본 연구에서는 SVC 명령어를 수행하여 SVC 예외를 발생시키는 SVC 호출 함수를 사용자 공간에 시스템 호출마다 구현하였다. SVC 호출 함수는 SVC 예외를 발생시키는 부분과 SVC 예외가 반환되었을 때 시스템 호출의 반환값이 있는 경우 그 값을 반환하는 부분으로 구성된다. 그림 1은 본 연구에서 구현한 SVC 호출 함수의 의사코드이다. 본 연구에서는 SVC 호출 함수를 C 언어와 어셈블러 (assembler)를 혼합하여 구현하였다.

그림 1의 3번 줄에서 SVC 명령어를 수행하여 SVC 예외를 발생시킨다. 이때 SVC 명령어의 인수로 시스템 호출에 할당된 번호를 전달한다. 이 번호는 SVC 예외 핸들러에서 해당하는 시스템 호출 함수를 시스템 호출 테이블에서 찾을 때 사용된다. 나중에 SVC 예외가 반환되면 4번 줄과 같이 시스템 호출의 반환값이 저장된 R0 레지스터의 값을 반환한다.

ARM 구조에서 함수가 호출되면 첫 번째 인수부터 네 번째 인수까지는 레지스터에 저장되고 나머지 인수들은 스택에 저장된다 [13]. 인수를 스택에 저장하는 것이 레지스터에 저장하는 것보다 더 오래 걸리기 때문에 함수가 여러 번 호출될 때마다 인수를 스택에 저장하느라 시간이 오래 걸린다. 본

```
1 __svc(void)
2 {
3     get psp value;
4     syscall_number = (psp[6])[-2];
5     load system_call_table[syscall_number]
6     push psp, lr using msp;
7     load r0-r3 from psp;
8     bl system_call_function;
9     pop psp, lr using msp;
10    store r0, psp[0];
11 }
```

그림 2. SVC 예외 핸들러의 의사코드
Fig. 2 Pseudo code for SVC exception handler

연구에서는 인수의 개수가 5개 이상인 시스템 호출의 경우에는 네 번째 인수부터 나머지 인수들을 구조체에 저장한 다음 이 구조체의 주소를 네 번째 인수로 전달하도록 하여 시스템 호출이 수행될 때 최대 4개의 인수만 전달되도록 하였다. 본 연구에서는 인수의 개수가 5개 이상인 시스템 호출들이 인수를 그대로 전달할 때와 구조체를 사용하여 인수를 전달할 때의 수행 시간을 측정하였으며, 그 결과 수행 시간이 평균 48 사이클 감소되었다.

Zephyr 커널에서 커널 서비스를 수행하려면 해당 커널 서비스를 위한 커널 객체를 초기화하고 이를 시스템 호출의 인수로 전달해야 한다. 커널 객체는 구조체 형태로 커널 공간에 정의되어 있기 때문에 공간 분리가 구현되었을 때 사용자 코드에서 커널 객체를 초기화하고 인수로 전달할 수 없다. 본 연구에서는 커널 객체 구조체 배열 형태인 커널 객체 풀을 커널에 객체마다 구현하여 이 문제를 해결하였다.

2. SVC 예외 핸들러

SVC 예외가 발생하면 하드웨어에 의해 SVC 예외 핸들러가 수행된다. 본 연구에서는 기존의 핸들러 코드를 제거하고 새로운 핸들러를 구현하였다. SVC 예외가 발생하면 동작 모드는 핸들러 모드로 변경되고 MSP가 가리키는 커널 스택이 사용된다. 그림 2는 새로 구현된 SVC 예외 핸들러의 의사코드이다. 본 연구에서는 SVC 예외 핸들러를 C 언어와 어셈블러를 혼합하여 구현하였다.

그림 2의 의사코드의 3-4번 줄에서 PSP 값을 로드하여 PSP가 가리키는 사용자 쓰레드가 사용 중인 스택에 저장된 시스템 호출 번호를 로드한다. 시스템 호출 번호는 사전 연구의 3절에서 설명한 예외가 발생했을 때 하드웨어가 현재 사용 중인 스택에 저장한 여러 레지스터들의 값 중에서 반환 주소의 첫 번째 바이트에 저장되어 있다 [9].

5번 줄에서 시스템 호출 번호를 시스템 호출 테이블의 인덱스 (index)로 사용하여 해당하는 시스템 호출 함수의 주소를 로드한다. 시스템 호출 테이블은 시스템 호출 함수들에 대한 함수 포인터를 원소로 가지는 배열이다. 시스템 호출 테이블을 사용함으로써 시스템 호출 함수의 주소를 로드할 때 항상 같은 시간이 소요된다.

함수 호출로 레지스터 값이 변경될 수 있기 때문에 6번 줄에서 PSP와 LR의 값을 MSP가 가리키는 커널 스택에 저장하고, 7번 줄에서 PSP가 가리키는 스택에서 시스템 호출 함수의 인수가 저장된 R0-R3 레지스터의 값을 로드한다. 그 다음에 8번 줄에서 시스템 호출 함수를 호출한다. 나중에 시스템 호출 함수가 반환되면 9번 줄에서 커널 스택에 저장했던 PSP와 LR의 값을 복구하고, 10번 줄에서 시스템 호출 함수의 반환값이 저장된 R0 레지스터 값을 PSP가 가리키는 스택에 저장한다. SVC 예외가 반환되면 PSP가 가리키는 스택에 저장되어 있는 레지스터들의 값이 하드웨어에 의해 자동으로 복구된다.

Ⅵ. 시험 및 평가

본 연구에서는 공간 분리가 구현된 Zephyr v1.8.0을 Cortex-M4 프로세서가 탑재된 NXP 사의 FRDM-K64F 보드에서 수행시켜 안전성과 에너지 소모량을 시험하였다.

1. 안전성 시험

본 연구에서는 공간 분리가 구현된 Zephyr 커널의 안전성을 시험하기 위해 다음과 같은 3개의 시험을 수행하였다.

- 시험 1: 사용자 코드가 실수로 커널 공간에 있는 임의의 주소에 대해서 쓰기 접근 (write access)을 했을 때 커널 수행에 문제가 발생하는지 확인함
- 시험 2: 사용자 코드가 의도적으로 커널 자료구조인 쓰레드를 표현하는 TCS (Thread Control Structure)를 수정했을 때 커널 수행에 문제가 발생하는지 확인함
- 시험 3: 사용자 코드가 의도적으로 커널 자료구조인 레디 큐 (ready queue)를 수정했을 때 커널 수행에 문제가 발생하는지 확인함

시험 1의 결과는 다음과 같다. 먼저 기존 Zephyr 커널의 경우 주소를 바꿔가면서 쓰기 접근을 하거나 함수 호출할 때마다 커널이 비정상적으로 수행되어 커널이 복구되지 않았다. 반면에 공간 분리가 구현된 Zephyr 커널의 경우 사용자 코드가 커널 공간에 접근하면 BusFault가 발생하여 현재 수행중인 쓰레드가 강제로 종료되고 다른 쓰레드가 수행되어 커널이 정상적으로 수행되었다.

시험 2와 시험 3의 결과는 다음과 같다. 먼저 기존 Zephyr 커널의 경우 사용자 코드에 의해 TCS와 레디 큐가 비정상적인 값으로 수정되었기 때문에 ISR 수행 중에 BusFault가 발생하여 커널이 복구되지 않았다. 반면에 공간 분리가 구현된 Zephyr 커널의 경우 사용자 코드가 TCS와 레디큐에 접근하면 BusFault가 발생하여 현재 수행중인 쓰레드가 강제로 종료되고 바로 다른 쓰레드가 수행되어 커널이 정상적으로 수행되었다.

본 연구에서 수행한 3개의 안전성 시험의 결과를 보면 사용자 코드가 실수 혹은 의도적으로 커널 코드를 수정하였을 때, 기존 Zephyr 커널은 커널이 비정상적으로 수행되어 복구되지 않았다. 반면에 공간 분리가 구현된 Zephyr 커널은 BusFault가 발생하여 현재 수행중인 쓰레드만 종료되고 커널은 정상적으로 수행되었다. 따라서 기존의 Zephyr 커널은 안전하지 않으며 공간 분리가 구현된 Zephyr 커널은 안전하다.

2. 에너지 소모 분석

본 연구에서는 공간 분리가 구현된 Zephyr 커널의 에너지 소모량을 분석하였다. 본 연구에서 사용하는 Cortex-M4 프로세서는 클럭 사이클 당 에너지 소모량이 고정적이고, 에너지 소모량이 클럭 사이클에 비례한다 [8, 11, 12]. 공간 분리가 구현되면 SVC 방식으로 인해 시스템 호출을 수행할 때에만 오버헤드가 발생하여 에너지 소모량이 증가되고, 응용 프로그램에서 시스템 호출을 얼마나 자주 사용하는지에 따라 증가되는 에너지 소모량이 달라진다. 이러한 이유로 본 연구에서는 클럭 사이클을 고려하여 에너지 소모량을 분석하였다.

표 1은 공간 분리가 구현되었을 때 Zephyr 커널의 주요 시스템 호출들의 오버헤드를 측정한 결과이다. 표 1의 결과에 따르면 시스템 호출이 함수 호출 방식에서 SVC 호출 방식으로 수정되면서 평균 60.5 사이클의 오버헤드가 발생하였다. 이 오버헤드는 SVC 예외처리 시 콘텍스트 저장 및 복구에서 발생하는 근원적인 오버헤드이다. 이 오버헤드

표 1. 시스템 호출의 오버헤드 측정 결과
Table 1. The result of system call overhead measurement

unit: clock cycles

System call	Function call	SVC call	Overhead
k_thread_create	324	428	104
k_thread_priority_set	243	293	50
k_mutex_lock	127	178	51
k_sem_take	44	88	44
k_mem_slab_alloc	38	96	58
k_pipe_get	370	426	56
Average	-	-	**60.5**

는 응용 프로그램에서 시스템 호출을 수행할 때마다 발생하여 그만큼의 전력을 더 소모하게 된다. Zephyr에서 제공하는 동기화 예제 프로그램은 약 30초 동안 수행될 때, 함수 k_sem_take가 60회 호출되며, k_sem_take 함수의 수행 시간 오버헤드가 44 사이클이므로 에너지 소모량이 0.00007% 증가한다. Zephyr 커널은 사물인터넷용 운영체제이며 Zephyr 커널 상에서 동작하는 응용 프로그램은 대부분의 시간을 대기상태에 있다가 센서나 다른 기기로부터 데이터를 전달받으면 데이터를 처리하고 다시 대기상태가 되는 것을 반복하기 때문에 시스템 호출을 자주 수행하지 않는다. 따라서 Zephyr 커널에서 시스템 호출로 인한 에너지 소모량의 증가는 크지 않을 것으로 판단된다.

Ⅶ. 결 론

사물인터넷용 운영체제는 기존 운영체제와 달리 적은 메모리 사용, 저 전력 기능, 실시간성, 멀티태스킹, 다양한 네트워크 기능, 안전성 등을 제공해야 한다. 2016년 2월 리눅스 재단에서 발표한 사물인터넷용 운영체제인 Zephyr 커널은 이러한 기능들을 갖추고 있지만 사용자 코드가 커널 코드와 같은 공간에서 수행되는 단일 공간 방식을 사용하기 때문에 사용자 코드에서 발생한 오류가 시스템에 치명적인 문제를 일으킬 수 있다. 본 연구에서는 이러한 문제를 해결하기 위하여 공간 분리 방식을 제안하고 이 방식을 Zephyr v1.8.0에 구현하였다. 공간 분리는 커널 공간과 사용자 공간을 분리하는 것이며 코드 분리, 커널 공간 보호, SVC 방식의 시스템

호출로 구성된다. 본 연구에서는 공간 분리가 구현된 Zephyr 커널에서 안전성을 시험하였으며 시험 결과 커널에 문제를 일으키는 사용자 코드의 수행이 기존 Zephyr 커널에서는 커널 수행에 영향을 주어 커널이 복구되지 않는 반면에 본 연구에서 제안하는 공간 분리가 구현된 Zephyr 커널에서는 커널 수행에 영향을 주지 않아 커널이 정상적으로 수행되었다. 또한 공간 분리가 구현된 Zephyr 커널의 에너지 소모량을 분석한 결과 공간 분리가 구현되면 시스템 호출을 수행할 때마다 오버헤드가 발생하여 이에 따라 에너지 소모량도 증가하지만 Zephyr 커널 상에서 동작하는 응용 프로그램에서 시스템 호출은 자주 수행되지 않으므로 시스템 호출로 인한 에너지 소모량의 증가는 크지 않을 것으로 판단된다.

References

[1] O. Hahm, E. Bacceli, H. Petersen, N. Tsiftes, “Operating Systems for Low End Devices in the Internet of Things: a Survey,” IEEE Internet of Things Journal, Vol. 3, No. 5, pp. 720-734, 2016.

[2] T.V. Chien, H.N Chan, T.N Huu, “A Comparative Study on Operating System for Wireless Sensor Networks,” Proceedings of IEEE Conference Publications, pp. 73-78, 2011.

[3] Zephyr Project Documentation, Available on https://docs.zephyrproject.org/1.11.0/, 2018

[4] H. Kim, H. Cha, “Towards a Resilient Operating System for Wireless Sensor Networks,” Proceedings of the 2006 USENIX Annual Technical Conference, pp. 103-108, 2006.

[5] K. Kim, I. Lee, “An Implementation of Mondriaan Memory Protection,” Proceedings of the KISS 2006 Fall Conference, Vol. 33, No. 2, pp. 276-280, 2006 (in Korean).

[6] E. Kim, J. Lim, B. Ko, D. Shin, “Separation of Kernel Space and User Space in Zephyr Kernel,” Proceedings of the Conference of Institute of Embedded Engineering of Korea 2017, Vol. 1, No. 1, pp. 353-356, 2017 (in Korean).

[7] ARM, ARMv7-M Architecture Reference Manual, ARM DDI 0403E.b, 2014.

[8] ARM, ARM Cortex-M4 Processor Technical Reference Manual, Revision: r0p1, ARM 100166_0001_00_en, 2015.

[9] J. Yiu, The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, 3rd Edition, Elsevier Inc., 2014.

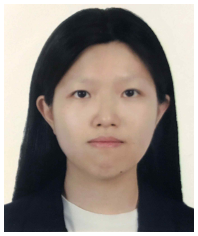
[10] NXP, FRDM-K64F Freedom Module User's Guide Rev 1, FRDMK64FUG, 2016.

[11] NXP, K64 Sub-Family Reference Manual, Rev 2, 2014.

[12] NXP, Kinetis K64F Sub-Family Data Sheet, Rev. 6, 2015.

[13] ARM, Procedure Call Standard for the ARM Architecture, ARM IHI0042F, 2015.

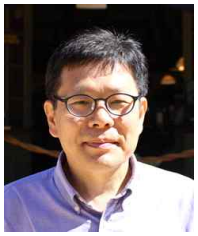
Eunyoung Kim (김 은 영)



She received B.E. degree and M.S degree in Computer Science from Sangmyung University in 2016 and 2018. Her research interests include real-time kernels, kernel protection, ARM architecture and embedded computing.

Email: eykim0322@naver.com

Dongha Shin (신 동 하)



He received the B.S. degree in Computer Engineering from Kyungpook National University in 1980, the M.S. degree in Computer Engineering from Seoul National University in 1982 and the Ph.D. degree in Computer Science from University of South Carolina in 1994. During 1982-1996, he stayed in ETRI as a technical staff to study expert systems, word processing systems, file systems and language processing systems. During 1997-current, he is a professor of Computer Science Department in Sangmyung University. His research interests include real-time kernels, kernel protection, embedded computing and compiler implementation.

Email: dshin@smu.ac.kr