운영체제 커널에서 문맥교환 구현 및 분석

(Implementation and Analysis of Context Switching on OS Kernel)

정 주 영[†], 신 동 하[‡*]
[†]상명대학교 일반대학원 컴퓨터과학과, [‡]상명대학교 ICT융합대학 컴퓨터과학과
(Jooyoung Jung, Dongha Shin)
([†]Department of Computer Science, Graduate School of Sangmyung University., [‡]College of ICT Convergence, Sangmyung University.)

Abstract : Context switching is necessary for kernel to support multithreading. Since a processor can run one thread at a time, the processor have to divide the time and switch the threads to run. When a kernel decides to run a different thread, it saves the current thread's context in the current thread's context save area. After this operation is performed, the new thread's context is restored from its save area and then resumes execution of the new thread's code. This process is called a context switch. Context switch time is pure overhead and it is highly dependent on operating systems. In this paper, we implement context switching on operating system kernel based on ARMv7 and measure the cycle timing of the kernel on a BeagleBone which has ARM Cortex-A8 processor. Furthermore, we analyze the cycle timing of the kernel in accordance with the instruction cycle timing described in ARM Cortex-A8 reference manual and compare it with FreeRTOS and uC/OS-II. As the result of comparison, we found the context switching implemented in this research require less cycle than other operating systems.

Keywords : Kernel, ARMv7, Context Switching, Scheduling, Cycle Timing

I. 서 론

대부분의 현대 운영체제는 멀티스레딩을 지원한다[1][3]. 멀티스레딩이란 여러 개의 스레드가 자원을 공유하기 위해 스레드를 관리, 전환하여 여러 개의 스레드가 동시에 실행되는 것처럼 보이게 하는 기술을 말한다[2][4]. 한 개의 프로세서가 한 순간에 수행할 수 있는 스레드의 개수는 한 개이기 때문에 프로세서는 주어진 스케줄링 방식에 따라 다음에 수행할 스레드를 선택하여 수행한다[1][2].

스케줄러가 다음에 실행시킬 스레드를 선택하면, 커널은 실행 중이던 스레드의 문맥을 저장하고 새로운 스레드의 문맥을 복구하는 작업을 수행한다[3]. 이 작업을 문맥교환이라고 한다. 여기서 문맥이란 스레드 수행에 관련된 데이터를 의미하며 프로세스 상태, 레지스터 값, 스택 포인터, 프로그램 카운터 등이 포함될 수 있다[2].

문맥교환이 일어나는 동안 시스템은 아무런 유용한 일을 하지 못하기 때문에 문맥교환 시간은 운영체제 커널에게는 순수 오버헤드이다[2]. 그 시간은 메모리 속도, 복사되어야 할 레지스터 수, 특수 명령어(한 개의 명령어로 여러 레지스터를 적재하고 저장하는 명령어) 존재 여부 등에 의해 좌우되므로 기계나 운영체제마다 다르다[2].

본 논문은 ARMv7[9] 기반의 운영체제 커널에서 문맥교환 기능을 구현하였다. 또한 구현한 코드의 수행 사이클을 ARM Cortex-A8[10] 매뉴얼에 따라 분석하고 ARM Cortex-A8 프로세서가 탑재된 BeagleBone[11][12] 상에서 성능을 측정하였다. 또한 분석 및 측정 결과를 바탕으로 기존 운영

* 교신저자(Corresponding Author) 신동하
※ 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. B0101-15-0644, 매니코어 기반 초고성능 스케일러블 OS 기초연구)

체제인 FreeRTOS[4][5], uC/OS-II[7][8]와 비교 분석하였다.

본 논문의 구성은 다음과 같다. 먼저 2장에서 문맥교환 구현에 필요한 사전지식을 설명하고 3장에서 본 연구에서 구현한 내용에 대해서 설명한다. 4장에서는 본 연구에서 구현한 문맥교환의 수행 사이클을 분석하고 측정한 결과와 FreeRTOS, uC/OS-II와 비교한 내용에 대하여 설명한다. 마지막으로 5장에서는 본 논문의 결론에 대하여 기술한다.

II. 본 론

이 장에서는 ARMv7 기반 운영체제 커널의 문맥교환 기능을 구현하기 위해 필요한 기본 지식인 ARM 레지스터 세트, 문맥교환의 기본 동작 및 문맥교환의 사례에 대해 설명한다.

1. ARM 레지스터 세트

ARM 아키텍처에서 레지스터 세트는 31개의 32 bit 범용 레지스터와 6개의 32 bit 상태 레지스터로 구성되는데 범용 레지스터는 여러 동작 모드에서 공유하여 사용하는 레지스터와 각 동작 모드별로 할당된 레지스터가 있다[9]. 동작 모드에 따라 16개(R0-R15)만 접근이 가능하다[9]. 또한 R13, R14, R15 레지스터는 특수 목적 레지스터로 각각 스택 포인터(SP), 링크 레지스터(LR), 프로그램 카운터(PC)로 사용된다[9]. 상태 레지스터는 1개의 CPSR(Current Program Status Register)와 5개의 SPSR(Saved Program Register)로 구성되며 CPSR은 현재 프로세서의 상태를, SPSR은 익셉션 발생 바로 전의 프로세서 상태를 저장하고 익셉션 모드별로 1개씩 존재한다[9].

2. 문맥교환 수행 과정

이 절에서는 문맥교환의 일반적인 실행 과정을 설명한다. 일반적으로 문맥교환은 타이머 인터럽트가 발생한 경우와 스레드가 문맥교환을 일으키는 시스템 콜을 호출한 경우에 일어난다[1]. 두 경우 모두 예외가 발생하여 익셉션 핸들러가 실행되고 익셉현 핸들러가 문맥교환을 수행한다. 문맥교환을 수행하기 위해서는 다음에 수행할 스레드를 선택하기 위한 스케줄링 단계를 거쳐야 하며 실제 문맥교환 작업인 문맥을 저장하고 복구하는 단계를 거치면 문맥교환이 완료된다. 본 논문에서는 문맥교환의 과정을 스케줄링 단계와 문맥 저장 및 복구 단계로 나누어 설명한다.

2.1 스케줄링

스케줄러는 문맥교환의 수행 여부를 결정하고 준비 상태에 있는 스레드 중에서 다음에 수행할 스레드를 선택한다. 이때 어떤 스레드를 선택할지는 운영체제가 사용하는 스케줄링 방식에 따라 결정된다[1]. 스케줄러가 다음에 수행할 스레드를 선택하면 선택된 스레드로 실제 문맥교환이 일어난다[1][4].

2.2 문맥 저장 및 복구

문맥교환으로 인해 수행을 멈추는 스레드가 나중에 수행을 재개하기 위해서는 이 스레드의 문맥을 메모리의 특정 공간에 저장해야 한다[2]. 일반적으로는 스택에 저장하며 스택 포인터는 해당 스레드의 TCB에 저장된다[1][3]. 그 다음 새로운 스레드의 TCB에 저장되어 있는 스택 포인터 값을 프로세서의 SP 레지스터에 복사하여 새로운 스레드의 문맥이 저장된 스택을 가리키도록 한다[1][3]. 저장되어 있는 문맥을 새로운 스레드의 스택으로부터 프로세서 레지스터에 로드하고 PC 값을 프로세서의 PC 레지스터에 복사하면 문맥교환이 완료되고 새로운 스레드의 수행이 재개된다[1][3].

3. 사례조사

이 절에서는 FreeRTOS 및 uC/OS-II의 문맥교환에 사용된 기술에 대해서 설명한다. 이 두 OS의 문맥교환은 4장에서 본 연구에서 구현한 문맥교환과 함께 비교한다.

2.1 FreeRTOS

FreeRTOS의 스케줄링 방식은 우선순위 기반 라운드 로빈 방식으로 각 태스크는 중복된 우선순위 값을 가질 수 있다[6]. 각 우선순위마다 이중 원형 연결 리스트로 구현된 레디 리스트를 두고 준비 상태인 태스크들을 관리한다[6]. 스케줄링 시 스케줄러에 의해 선택된 스레드를 레디 리스트에서 제거하지 않으며 포인터로 마지막으로 선택된 스레드를 표시하여 다음 스케줄링에 사용한다. FreeRTOS는 모든 태스크마다 태스크 스택이 1개 있으며 여기에 문맥을 비롯한 태스크의 수행에 필요한 데이터를 저장한다.

2.2 uC/OS-II

uC/OS-II의 스케줄링 방식은 우선순위 스케줄링 방식으로 각 태스크는 유일한 우선순위 값을 가지며 준비 상태의 태스크들을 두 개의 변수로 구성된 준비 리스트에 해당 우선순위의 태스크가 준비 상태임을 표시하여 관리한다[8]. uC/OS-II도 FreeRTOS와 마찬가지로 모든 태스크마다 태스크 스택이 1개씩 있으며 그 쓰임도 비슷하다.

Ⅲ. 설계 및 구현

본 연구에서는 문맥교환 기능을 구현하기 위해 두 개의 함수를 구현하였다. 두 함수는 앞서 설명한 스케줄링, 문맥 저장 및 복구 기능을 각각 수행한다. 이 장에서는 두 함수를 구현한 내용에 대해서 설명한다.

1. 스케줄링

본 연구에서 설계한 스레드 스케줄링 방식은 우선순위 기반 라운드로빈 방식이다. 이 스케줄링 방식은 기본적으로 우선순위 스케줄링 방식을 따르되 같은 우선순위를 가진 스레드들에 대해서는 라운드로빈 방식으로 스케줄링 한다. 본 연구에서 구현한 커널에서 우선순위 값의 범위는 0부터 31까지 이며 각 스레드는 중복된 우선순위 값을 가질 수 있다. 따라서 우선순위마다 레디 큐를 할당하고 준비 상태인 스레드는 스레드의 우선순위에 해당하는 레디 큐에서 관리한다. 스케줄러는 가장 높은 우선순위의 레디 큐를 찾고 해당 레디 큐에 있는 첫 번째 스레드를 다음에 수행할 스레드로 선택한다.

2. 문맥 저장 및 복구

본 연구에서 구현한 문맥 저장 및 복구 함수는 현재 스레드와 다음에 수행할 스레드의 TCB 포인터를 매개변수로 갖는다. 이 TCB 내에는 유저 스택과 커널 스택을 가리키는 스택포인터 필드가 있으며 유저 스택은 사용자 프로그램을 수행할 때 사용되고 커널 스택은 예외 처리, 문맥교환 등 커널 기능을 수행할 때 사용된다.

문맥 저장 및 복구 함수를 호출하면 CPSR, R0 - R12, LR, LR 값을 현재 스레드의 커널 스택에 차례로 저장한 후 SP 값을 TCB에 저장하여 문맥을 저장한다. 문맥 저장 후 프로세서의 SP 레지스터에 다음에 수행할 스레드의 TCB로부터 커널 스택 포인터 값을 복사하여 스택 포인터를 변경한다. 그 다음, 커널 스택에 저장되어 있던 문맥을 CPSR, R0-R12, LR, PC에 로드하여 문맥을 복구한다. 문맥을 복구하고 나면 프로세서는 이전에 스레드가 수행을 멈추기 전의 상태로 돌아가서 스레드의 수행을 재개하며 문맥교환이 완료된다.

Ⅳ. 분석 및 측정

이 장에서는 구현한 문맥교환 코드의 성능을 측정하기 위하여 구현한 코드의 수행 사이클을 ARM Cortex-A8 사이클 타이밍[10]에 따라 분석하고 본 연구에서 개발 중인 커널인 ARM-Kernel에 적용하여 BeagleBone[11][12] 상에서 측정한 결과에 대하여 기술한다. 또한 분석 및 측정한 결과를 바탕으로 기존 OS인 FreeRTOS[4][5] 및 uC/OS-II[7][8]와 비교한다.

1. 분석

본 연구에서는 분기 예측기가 enable되어 있어 분기 명령으로 인한 파이프라인 플러시 지연은 없으며 코드 수행에 필요한 모든 데이터가 캐시에 있다는 가정 하에서 ARM-Kernel, FreeRTOS 및 uC/OS-II의 문맥교환 코드의 수행 사이클을 분석하였다.

표 1. 문맥교환 코드 사이클 분석 결과
Table 1. The result of analysis for context switching code cycle

	ARM-Kernel	FreeRTOS	uC/OS-II
스케줄링	70	34	63
문맥 저장 및 복구	40	73	56
합계	110	107	119

단위: 사이클

표 1은 분석 결과를 비교한 표이다. 스케줄링 단계에서 FreeRTOS의 수행 사이클이 가장 적고 ARM-Kernel에서 가장 많은데 이는 스케줄링을 어떻게 하는가에 따라 다르다. 스케줄링 시 FreeRTOS는 사전지식에서 설명한 바와 같이 스케줄러에 의해 선택된 스레드를 레디 리스트에서 제거하지 않는 반면 ARM-Kernel은 레디 큐에서 제거하기 때문이다.

문맥 저장 및 복구 단계에서는 ARM-Kernel의 수행 사이클이 다른 두 OS에 비해 적은데 그 이유는 먼저 ARM-Kernel은 문맥을 저장하고 복구하는 동작만 하는데 반해 FreeRTOS와 uC/OS-II는 변수 값을 변경하거나 다른 함수를 호출하기 때문이다. 또한 FreeRTOS와 uC/OS-II의 추가적인 동작을 제외한 사이클을 비교했을 때에도 ARM-Kernel에서

가장 적은 사이클이 필요한 것으로 분석하였는데 그 이유는 ARM-Kernel은 문맥 저장 및 복구 함수의 매개변수로 교환되는 두 스레드의 TCB 주소를 전달하는데 반해, 다른 OS에서는 TCB를 접근하기 위해서는 TCB 주소를 레지스터에 로드하는 작업이 필요하기 때문이다. TCB 주소를 1회 로드하는데 필요한 사이클은 2 cycle이다. 따라서 FreeRTOS는 TCB 주소를 2회 로드하는데 4 cycle, 문맥을 저장할 때 스택 포인터를 변경하는데 12 cycle 그리고 NOP 명령어 사용으로 3 cycle이 추가적으로 필요한 것으로 분석했다. uC/OS-II의 경우 TCB 주소를 2회 로드하는데 4 cycle, LDM 명령어의 반복 사용으로 인해 2 cycle이 더 필요한 것으로 분석했다.

2. 측정

본 논문에서는 성능 측정을 위해 ARMv7에서 제공하는 CP15의 성능 모니터링 기능[9]을 사용했다.

표 2. 문맥교환 성능 측정 결과
Table 2. The result of measurement for context switching performance

	ARM-Kernel (최소/평균)	FreeRTOS (최소/평균)	uC/OS-II (최소/평균)
스케줄링	71/71	35/49	74/74
문맥 저장 및 복구	76/76	126/126	108/115
합계	147/147	161/175	182/189

단위: 사이클

표 2는 측정 결과를 비교한 표이다. 표 2에서 ARM-Kernel과 기존 커널을 비교한 결과 스케줄링 단계에서는 더 많은 사이클이 소요되었으나 분석과 마찬가지로 문맥 저장 및 복구 단계에서 가장 적은 사이클이 소요된다는 것을 확인할 수 있었다.

V. 결 론

본 논문에서는 ARMv7구조 운영체제 커널에서 문맥 교환 기능을 구현한 방법에 대하여 설명하였으며 구현한 코드 수행 사이클을 분석하고 본 연구에서 개발 중인 커널에 적용하여 ARM Cortex-A8 프로세서가 탑재된 BeagleBone 상에서 측정하였다. 또한 분석 및 측정한 결과를 바탕으로 기존 OS인 FreeRTOS와 uC/OS-II와 비교하였다. 비교 결과 스케줄링 단계에서는 기존 운영체제보다 많은 사이클이 소요되었으나 문맥 저장 및 복구 단계에서는 가장 적은 사이클이 필요함을 확인하였다.

참 고 문 헌

[1] Thomas Anderson, and Michael Dahlin, Operating Systems Principles & Practice, 2th edition, 2014.

[2] Abraham Silberchatz, Peter B. Galvin and Greg Gagne, Operating System Concepts, 9th edition, WILEY, 2013.

[3] Douglas Comer, Operating System Design: The Sinu Approach Linksys Version, CRC Press, 2012.

[4] FreeRTOS, http://www.freertos.org.

[5] Richard Barry, The FreeRTOS Reference Manual API Functions and Configuration Options, Real Time Engineers Ltd., 2014.

[6] Amy Brown, The Architecture of Open Source Applications, Volume 2, Lulu.com, 2008.

[7] uC/OS-II, http://micrium.com/rtos/ucosii.

[8] Jean J. Labrosse, MicroC/OS-II The Real Time Kernel, 2th edition, CMP Books, 2002.

[9] ARM Limited, ARM Architecture Reference Manual, ARMv7-A and ARMv7-R Edition, ARM DDI 0406C.c, 2014.

[10] ARM Limited, Cortex-A8 Technical Reference Manual, ARM DDI 0344K, 2010.

[11] Beaglebone, http://beagleboard.org/bone.

[12] G Coley, BeagleBone Rev A6 System Reference Manual Rev 0.0, beagleboard.org, 2012.