운영체제 커널에서 Driect Thread Switching을 사용한 IPC 구현 및 분석

(IPC Implementation and Analysis Using Direct Thread Switching on Operating System Kernel)

김 은 영†, 신 동 하†*
†상명대학교 ICT융합대학 컴퓨터과학과
(Eunyoung Kim, Dongha Shin)
(†Department of Computer Science, College of ICT Convergence, Sangmyung University)

Abstract : Microkernel operating system implements basic mechanisms in the kernel and other policies as service programs. Since service programs are implemented in its own address space, they communicate each other via inter-process communication(IPC). IPC is a mechanism that transfers a data between source thread and destination thread. Since IPC is performed very frequently in microkernel, IPC performance is very important. However when there are many threads of ready state, IPC performance is decreased because many threads need to execute before IPC destination thread execute. Many studies have been proposed to improve the IPC performance and direct thread switching is known to be effective in solving this problem. Direct thread switching can improve IPC performance because destination thread execute immediately without scheduling after source thread sends a data to destination thread. In this paper, we implemented synchronous IPC for microkernel and used direct thread switching to improve IPC performance. Through the results of IPC performance measurement of uC/OS-II, FreeRTOS and the kernel implemented in this paper, we confirmed that IPC performance is not good if we don't use direct thread switching. Also we analyzed the IPC performance differences when using direct thread switching and not using it.

Keywords : microkernel, ARM, IPC, direct thread switching

I. 서 론

마이크로커널 운영체제는 기본 메카니즘은 커널 내에서 구현하고 나머지 정책들은 커널 밖 서비스 프로그램으로 구현하는 운영체제이다. 마이크로커널 운영체제는 유연성(flexibility), 안정성(safety), 모듈성(modularity)의 장점이 있다[1]. 마이크로커널에서 서비스 프로그램들은 각각의 주소공간에서 수행되기 때문에 IPC를 통해 서로 통신한다[1]. IPC(Interprocess Communication)란 스레드와 스레드 사이에서 데이터를 전달해주는 메카니즘이며 마이크로커널에서 매우 자주 수행되기 때문에 IPC 성능은 매우 중요하다. 그런데 만약에 ready상태인 스레드가 많이 있으면, 이 스레드들이 모두 수행을 마쳐야 수신자 스레드가 수행되기 때문에 IPC 성능이 감소한다. IPC 성능을 향상시키기 위한 연구가 많이 진행되어 왔는데 그 중 direct thread switching[2][4]이 위의 문제를 해결하는데 효과적이라고 알려져 있다. Direct thread switching을 사용하면 송신자 스레드가 데이터를 보낸 뒤에 스케줄러가 호출되지 않고 바로 수신자 스레드가 수행되기 때문에 IPC 성능을 높일 수 있다.

본 논문에서는 마이크로커널용 synchronous IPC를 구현하였고 성능향상을 위해 direct thread switching을 사용하였다. 본 논문에서 구현한 IPC의 성능과 uC/OS-II[6][7]와 FreeRTOS[8][9]의 IPC 성능을 측정한 결과를 통해 direct thread switching을 사용하지 않으면 IPC 성능이 좋지 않

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

다는 것을 확인하였고 direct thread switching을 사용할 때와 사용하지 않을 때의 IPC 성능의 차이가 발생하는 이유를 분석하였다.

본 논문의 2장에서는 synchronous IPC, direct thread switching과 ARM의 performance counting을 설명한다. 3장에서는 본 논문에서 구현한 IPC 대해 기술한다. 4장에서는 다양한 시스템에서 IPC 성능을 측정한 결과를 기술하고, direct thread switching을 사용할 때와 사용하지 않을 때의 IPC 성능차이를 분석한다. 5장에서는 본 논문의 결론을 기술한다.

II. 사 전 지 식

이 장에서는 Synchronous IPC와 direct thread switching에 대하여 설명한다. 그리고 4장에서 IPC 수행 사이클 수를 측정할 때 사용하는 ARM의 performance monitor[5]기능을 사용하는 방법에 대하여 설명한다.

1. Synchronous IPC

IPC가 동기적(synchronous)이면 수신자 스레드가 IPC를 받을 때까지 송신자 스레드는 대기상태에 머물러 있다가 수신자 스레드가 IPC를 받으면 준비상태로 돌아간다[3]. 그리고 Synchronous IPC는 IPC 성능을 높이기 위한 기법인 temporary mapping, lazy scheduling, direct thread switching을 구현하는데 필요한 전제조건이다[4]. L4계열의 커널들은 synchronous IPC를 기본적으로 지원하면서 asynchronous notification 기능도 제공하고 있다[4].

2. Direct Thread Switching

일반적으로 수행중인 스레드가 수행을 마치면 스케줄러가 호출되어 다음에 수행할 스레드를 레디 큐에서 선택하고 선택된 스레드가 수행된다. IPC를 수행하는 동안에도 스케줄러가 다음에 수행할 스레드를 찾는다. 그런데 만약 레디 큐에 스레드가 많이 있다면 그 스레드들이 모두 수행된 후에 수신자 스레드가 수행되기 때문에 데이터 전달이 늦어지게 된다. 이를 해결하는 방법으로 direct thread switching[2][4]이 있다. Direct thread switching을 사용하면 송신자 스레드가 수행을 마쳤을 때 스케줄러가 호출되지 않고 수신자 스레드가 바로 수행된다. 데이터를 전달하는 동안 다른 스레드가 수행되지 않기 때문에 데이터 전달이 빨라져서 IPC 성능이 향상된다. 그런데 direct thread switching을 사용하면 스레드의 우선순위를 무시하고 바로 수신자 스레드를 수행하기 때문에 문제가 발생할 수 있다. L4 계열의 커널에서는 우선순위에 문제가 없으면 direct thread switching을 사용하지만 있으면 일반적인 스케줄러를 호출한다[4].

3. ARM Performance Monitor

ARMv7에서는 성능 측정을 위해 Coprocessor15에서 system performance monitor 기능을 제공한다[5]. performance monitor는 1개의 사이클 카운터와 최대 31개의 이벤트 카운터로 구성된다[5]. 본 연구에서는 수행 사이클 수를 측정하기 위해 cp15 레지스터들 중에서 PMCR, PMCNTENSET, PMOVSR, PMCCNTR, PMUSERENR, PMIMTENCLR을 사용한다. 위의 레지스터들을 사용하여 어떤 동작의 성능을 측정하려면 먼저 위의 레지스터들을 초기화 하고 성능을 측정하고자 하는 동작의 시작 직전과 종료 직후의 PMCCNTR 값을 읽는다. 종료 직후의 PMCCNTR 값에서 시작 직전의 PMCCNTR값을 빼면 그 동작을 수행하는데 걸린 사이클 수를 측정할 수 있다.

III. 구 현

이 장에서는 IPC를 구현하기 위해 필요한 TCB의 여러 필드들과 IPC 송신과 수신을 위한 시스템 콜을 설명한다. 그리고direct thread switching 구현을 설명한다.

1. Thread Control Block(TCB)

본 연구에서는 IPC 설계에 마이크로커널의 주요 자료 구조인 TCB에 저장된 여러 필드들이 사용된다. TCB는 커널 자료 구조이므로 커널 내에서 효율적으로 접근이 가능하다. TCB에 저장된 필드 중 IPC 수행 시 사용되는 필드들은 다음과 같다.

1.1 tid

스레드의 id를 저장하는 필드이다. IPC를 수행할 때 송신자 스레드와 수신자 스레드를 구분하기 위해 사용된다.

1.2 state

스레드의 상태를 저장하는 필드이다. 스레드의 상태에는 READY, RUN, WAITING이 있다.

1.3 message

스레드 사이에서 IPC를 통해 주고 받을 메시지

를 저장하는 필드이다. 송신자 스레드가 보낸 메시지가 수신자 스레드의 TCB의 이 필드에 저장된다.

1.4 rbtid, sbtid, rwtid

각각의 경우에 따라 tid를 저장하는 필드들이다. rbtid에는 수신자 스레드가 송신자 스레드에게 IPC 수신 호출을 하고 블럭되면 송신자 스레드의 tid가 저장된다. sbtid에는 송신자 스레드가 수신자 스레드에게 IPC 송신 호출을 하고 블럭되면 수신자 스레드의 tid가 저장된다. rwtid에는 수신자 스레드가 특정한 스레드가 아닌 어떤 스레드에게 IPC 송신 호출을 하고 블럭되어 있으면 송신자 스레드가 송신 호출을 할 때 수신자 스레드에게 송신자 스레드의 tid를 알려주기 위하여 송신자 스레드의 tid가 저장된다.

2. 시스템콜

본 연구에서는 IPC의 송신과 수신을 위해 다음과 같은 시스템콜을 구현하였다.

1.1 IPC 송신 시스템콜

현재 수행중인 스레드가 다른 스레드에게 메시지를 보내기 위해 호출한다. 만약 현재 수행중인 스레드에게 메시지를 요청한 스레드가 있으면 시스템콜은 요청한 스레드를 레디 큐에 추가하고 메시지를 보낸 후 종료된다. 메시지를 요청한 스레드가 없으면 시스템콜은 인수로 받은 스레드에게 메시지 송신을 요청하고 현재 수행중인 스레드를 wait상태로 만든다.

1.2 IPC 수신 시스템콜

현재 수행중인 스레드가 다른 스레드에게서 메시지를 받기 위해 호출한다. 만약 현재 수행중인 스레드에게 메시지를 송신한 스레드가 있으면 송신한 스레드의 TCB에서 메시지를 읽어오고 송신한 스레드의 상태를 ready로 바꾼다. 메시지를 송신한 스레드가 없으면 시스템콜은 인수로 받은 스레드에게 메시지 수신을 요청하고 현재 수행중인 스레드를 wait 상태로 만든다.

3. Direct Thread Switching

본 논문에서는 direct thread switching을 IPC 수신 시스템콜에서 수행되도록 구현하였다. 원래 IPC 수신 시스템콜에서 현재 수행중인 스레드에게 메시지를 요청한 스레드가 있으면 시스템콜은 메시지를 전송하고 수신자 스레드를 레디 큐에 추가하고 종료됐지만 direct thread switching을 사용하면 수신자 스레드가 레디 큐에 추가되고서 바로 수신자 스레드가 수행된다. 현재 수행중인 스레드를 레디 큐에 넣고 수신자 스레드를 레디 큐에서 제거하여 현재 수행중인 스레드의 남은 타임 슬라이스동안 수신자 스레드가 수행된다.

IV. 측정 및 분석

이 장에서는 uC/OS-II, FreeRTOS와 본 논문에서 구현한 ARM-Kernel-DTS에서 측정한 IPC 성능을 비교해보고 direct thread switching을 사용할 때와 사용하지 않을 때의 IPC 성능차이가 발생하는 이유를 분석한다.

1. 각 시스템에서의 IPC 성능 측정 결과

본 논문에서 구현한 direct thread switching을 사용하는 ARM-Kernel-DTS, direct thread switching을 사용하지 않는 ARM-Kernel. uC/OS-II[6][7] 및 FreeRTOS[8][9]에서 IPC 성능을 각각 측정하였다. 성능 측정 프로그램은 ARM Cortex-A8기반의 beaglebone에서 수행시켰으며 성능을 측정하기 위해 ARM의 performance monitor register[5]를 사용하였다.

표 1. 각 시스템에서의 IPC 성능 측정 결과
Table 1. The results of ipc performance measurement in each system

시스템	uC/OS-II	FreeRTOS	ARM-kernel	ARM-kernel-DTS
busy_wait	-	260×10^6	297×10^6	1392
sleep&busy_wait	92×10^6	116×10^6	120×10^6	1015
sleep	650	2500	333	299

단위: 사이클

위의 표 1은 각 시스템의 IPC 성능을 측정한 결과를 비교한 표이다. 측정값은 IPC 송신 시스템콜을 호출하기 바로 직전부터 IPC 수신 시스템콜의 수행을 마치고 난 바로 직후까지 수행된 사이클 수이며 값이 작을수록 성능이 좋음을 의미한다. busy_wait과 sleep은 송신자와 수신자 스레드를 제외한 나머지 스레드들이 수행하는 함수이다. busy_wait은 반복문을 수행함으로써 CPU를 사용하게 하는 함수이고 sleep은 스레드의 상태를 주어진 tick만큼 wait상태로 유지하는 함수이다.

IPC 성능 측정은 3가지 경우로 나눠서 측정했는

데 각 경우마다 레디 큐에 추가되는 스레드의 개수가 다르다. busy_wait만 수행할 때에 가장 많이 추가되며 sleep만 수행할 때 가장 적게 추가된다. 표 1의 측정 결과에 의하면 본 연구에서 구현한 direct thread switching을 사용하는 ARM-Kernel-DTS의 IPC 성능이 모든 경우에서 더 좋은 성능을 보였다.

2. Direct thread switching을 사용할 때와 사용하지 않을 때의 IPC 성능 차이 분석

위에서 측정한 결과를 통해 direct thread switching을 사용할 때 IPC 성능이 더 좋은 것을 확인하였다. 다음의 표 2는 direct thread switching사용 여부만 다른 두 개의 커널에서 스레드 개수에 따른 IPC 성능을 측정한 결과를 비교한 표이다.

표 2. Direct thread switching을 사용할 때와 사용하지 않을 때의 IPC 성능측정 결과

Table 2. The results of IPC performance measurement when using direct thread switching and not using

스레드 개수	ARM-kerenl	ARM-kernel -DTS
0개	456	378
30개	170×10^6	843
60개	340×10^6	1025
90개	510×10^6	1104
120개	680×10^6	1146
150개	805×10^6	1128
180개	1020×10^6	1153
200개	1133×10^6	1186

단위: 사이클

표 2에서 스레드의 개수는 송신자와 수신자 스레드를 제외한 나머지 스레드들의 개수를 의미한다. 표 2의 측정 결과에 의하면 direct thread switching을 사용하지 않으면 스레드 개수가 증가할수록 성능이 낮게 측정되었지만 direct thread switching을 사용하면 스레드 개수가 증가해도 성능에 영향을 주지 않았다. Direct thread switching 사용 여부에 따라 IPC 성능의 차이가 발생하는 이유는 다음과 같다. Direct thread swiching을 사용하지 않으면 레디 큐에 있는 스레드들이 모두 수행을 마쳐야 수신자 스레드가 수행될 수 있다. 그래서 스레드의 개수가 증가할수록 IPC 성능이 낮아진다. 반면에 direct thread switching을 사용하면 스레드의 개수가 증가하여 레디 큐가 길어져도 송신자 스레드가 데이터를 보낸 후에 스케줄러가 다음에 수행할 스레드를 선택하지 않고 수신자 스레드가 바로 수행되기 때문에 데이터 전달이 빨라져 IPC 성능이 향상된다.

V. 결 론

본 논문에서는 마이크로커널용 synchronous IPC를 구현하였고 성능향상을 위해서 direct thread switching을 사용하였다. Direct thread switching을 사용하지 않는 다른 커널들과 IPC 성능을 비교해 본 결과 direct thread switching을 사용했을 때의 IPC 성능이 월등히 좋다는 것을 확인하였고 direct thread switching을 사용할 때 성능이 좋아지는 이유에 대해서 분석하였다.

참 고 문 헌

[1] Jochen Liedtke, Toward Real Microkernels, Communications of the ACM 39(9), 1997.

[2] Jochen Liedtke, Improving IPC by kernel design, In Proceedings of the 14th ACM Symposium on Operating System Principles, 1993.

[3] Trent Jaeger, Jonathon E. Tidswell and Alain Gefflaut, Synchronous IPC over Transparent Monitors, Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System, 2000.

[4] Kevin Elphinstone and Gernot Heiser, From L3 to seL4:What Have We Learnt in 20 Years of L4 Microkernels?, In Proceedings of the 24th ACM Symposium on Operating Systems Principles, 2013.

[5] ARM Limited, ARM Architecture Reference Manual, ARM DDI 0406C.c, Chapter C12 The Performance Monitors Extension, 2014.

[6] J.Labrosse, MicroC/OS-II: The Real-Time Kernel, 2nd Edition, CMP Books, 2002.

[7] MicroC/OS-II, www.micrium.com, 2015.

[8] Real Time Engineers Ltd., FreeRTOS Reference Manual - API Functions and Configuration Options, 2014.

[9] FreeRTOS, www.FreeRTOS.org, 2015.