(사) 대 한 임 베 디 드 공 학 회
2014년도 추계학술대회 학술발표 논문집

2014 대한임베디드공학 추계학술대회

▣ 일 시 : 2014. 11. 13 ~ 15
▣ 장 소 : 제주 KAL 호텔
▣ 주 최 : (사)대한임베디드공학회
▣ 주 관 : 한국전자통신연구원, 대구경북과학기술원,
(재)경북IT융합산업기술원, (재) 경북차량용임베디드기술연구원
영남대학교정보통신연구소
▣ 후 원 : 언맨드솔루션, 레이디오펄스, 슈어소프트테크, 이노피아테크,
젠토, 코어테크놀로지, 네이버시스템즈, 테슬라시스템

(사) 대한임베디드공학회
Institure of Embedded Engineering of Korea

Interactive Session 2

좌 장 : 하옥균 박사 (경상대), 홍원기 교수(대구대)

L4 마이크로커널의 주요 기술 분석

(Analysis of Main Technologies in L4 Microkernel)

황 병 주†, 김 강 호‡, 신 동 하#
†상명대학교 일반대학원 컴퓨터과학과, ‡한국전자통신연구원 SW기초연구센터, #상명대학교 소프트웨어대학 컴퓨터과학과
(Byung-Ju Hwang, Kang Ho Kim, Dongha Shin)
(†Department of Computer Science, Graduate School of Sangmyung University, ‡Basic Research Center for SW, ETRI, #Department of Computer Science, College of Computer Software and Media Technology, Sangmyung University)

Abstract : It is expected that many-core systems with thousands cores will appear in the near future. In this paper, we analyze the technologies of L4 microkernel family which is succeeded in the field of microkernel for 20 years assuming that the future operating systems for many-core systems will be based on microkernel technologies. The success of L4 microkernel is due to the high-performance IPC which is developed by Liedtke in 1993. In order to achieve the high-performance IPC, Liedtke applied a variety of technologies such as synchronous IPC, direct transfer using temporary mapping, lazy scheduling and direct thread switch. These technologies have been changed and evolved to adapt many different environments for 20 years. L4 microkernel technologies that have been studied in this paper will be applied to design and develop a new operating system for many-core systems.

Keywords : Many-core system, Microkernel, L4

Ⅰ. 서론

가까운 미래에 수천 개의 코어를 가진 매니코어 시스템이 등장할 것으로 예측된다. 매니코어 기술은 메모리 속도, ILP, 전력 소비 등의 한계를 극복하기 위하여 필연적으로 등장하고 있다[1]. 매니코어 기술의 많은 장점에도 불구하고 현재의 운영체제는 확장성(scalability)이 부족하여 매니코어 시스템에 적합하지 않을 것으로 예측된다[2]. 최근 매니코어 시스템 혹은 유사 시스템의 운영체제로 HeliOS[3], Corey[4], Barrelfish[5], FOS[6], Tessellation[7] 등이 제안되었다. 이들 운영체제는 현재 널리 사용되는 모놀리식 커널 기반 운영체제와는 거리가 있다. 본 연구에서는 매니코어 시스템의 운영체제가 마이크로커널 기반의 운영체제가 될 것으로 추측하면서 마이크로커널 운영체제 분야에서 성공을 거둔 L4 계열 마이크로커널[8]의 최근 20년간의 기술을 분석하였다.

※ 본 연구는 미래창조과학부의 SW컴퓨팅산업원천기술개발 사업의 일환으로 수행하였음 (14-824-09-011, 매니코어 기반 초고성능 스케일러블 OS 기초연구).

마이크로커널 운영체제는 IPC, 스레드, 예외 처리, 메모리 공간 등 운영체제의 최소 기능만 특권 모드에서 실행하고, 가상 메모리 관리, 파일 시스템, 장치 관리 등과 같은 기능은 사용자 모드에서 실행하는 것이 원칙이다. 이 최소(minimality)의 원칙으로 인해 마이크로커널 운영체제는 모듈성(modularity), 유연성(flexibility), 보안성(security) 등의 장점을 제공한다[9]. 이러한 장점에도 불구하고 Mach[10]와 같은 초기 마이크로커널은 실제 성능이 기존 모놀리식 커널보다 약 30% 느려서[11] 상업적으로 실패했다. 그 후 1993년 Liedtke는 Mach 마이크로커널 대비 IPC 속도를 약 20배 향상시킨 L4 마이크로커널을 공개하면서 성능에 대한 기존 오해를 풀 수 있었다[12]. Liedtke는 고성능의 IPC를 구현하기 위하여 동기식 통신, 일시적 매핑을 사용한 직접 전송, lazy 스케줄링, 직접 쓰레드 전환 등과 같은 다양한 기술을 적용하였다. 그 이후 이들 기술은 20년간 여러 환경에 적응하기 위하여 변화 발전하였다. 본 논문은 그 동안 연구 개발된 L4 계열 마이크로커널의 기술들을 분석하였

다. 본 논문에서 분석된 L4 마이크로커널 기술은 본 연구팀이 수행 중인 매니코어 시스템용 운영체제 개발에 적용될 예정이다.

본 논문의 2장에서는 마이크로커널 성능의 주된 병목 지점인 IPC의 성능 향상 기술에 대해 분석하고, 3장에서는 스레드 관리, 4장은 예외 처리, 5장에서는 메모리 공간, 6장에서는 최적화에 대해 분석한다. 그리고 7장 결론에서는 본 논문의 의의 및 앞으로의 연구 계획에 대하여 기술한다.

II. IPC

마이크로커널에서 IPC는 운영체제가 제공하는 서비스 사이는 물론이고 응용프로그램과 서비스 간에도 빈번히 사용되므로 매우 고성능이어야 한다. Liedtke의 L4 커널은 동기식(synchronous) IPC를 사용하여 고성능을 이룰 수 있는 많은 기법을 얻어냈다. 동기식 IPC는 비동기식(asynchronous)에 비해 개념적으로나 구현에 있어서 비교적 간단하고, 커널 버퍼링을 사용하지 않으므로 복사 비용이 적다. Liedtke는 동기식 IPC 성능 향상에 사용할 수 있는 구현 기술을 제안 했는데, 일시적 매핑을 사용한 직접 전송, lazy 스케줄링, 직접 스레드 전환 등이 그것이다[12].

일반적으로 메시지 전송에는 두 번의 복사(sender 영역에서 커널 영역, 커널 영역에서 receiver 영역)가 필요하다. 하지만 일시적 매핑을 사용한 직접 전송은 메시지 복사를 한 번으로 줄여 성능을 높이는 기술이다. 커널은 복사해야할 내용이 있는 소스 영역을 커널 영역으로 복사하기 전에 최종 복사될 영역을 커널 영역에 잠시 매핑 시켜 통신창(communication window)을 만들고 한 번의 복사로 완료되게 한다.

동기식 IPC 모델에서는 한 타임 슬라이스 내에서도 해당 스레드의 상태가 실행(run)과 블록(block) 사이에서 자주 바뀌므로 이를 반영하기 위하여 준비 큐(ready queue)의 수정이 빈번히 일어난다. 큐 삽입에는 적어도 7번, 삭제에는 4번의 로드(load)와 스토어(store)가 일어나고, 4번 정도의 TLB 미스(miss)가 함께 발생하므로 비용이 크다. Lazy 스케줄링은 고비용인 큐 삽입과 삭제를 IPC 작동 시점에서 스케줄러 작동 시점으로 미루어 평균(average case) 성능을 높이는 기술이다[12].

동기식 IPC에서 메시지를 보내는 스레드는 일반적으로 메시지를 보낸 후 블록 되고, 커널은 스케줄러를 작동시켜 다음으로 실행할 스레드를 찾는다. 반면에 직접 스레드 전환은 커널이 스케줄러를 작동시켜야 할 시점에 메시지를 받는 스레드를 바로 실행시켜 IPC 응답성을 높이는 기술이다.

앞에서 설명한 기술들은 동기식 IPC의 성능 향상을 위한 기술이다. 하지만 동기식 IPC만 제공하면 스레드 하나에서 프로그래밍 할 수 있는 일을 여러 스레드에서 프로그래밍 해야 하는 경우가 생긴다[8]. 그래서 최근 L4 계열에서는 비동기식 메커니즘도 추가하는 추세이다. 이를테면 L4-embedded는 비동기식 알림(notification)을 추가했고, seL4는 비동기식 endpoint를 추가했으며, NOVA는 동기식 IPC에 계수 세마포어(counting semaphore) 기능을 추가하고, Fiasco.OC는 동기 IPC에 가상 IRQ를 추가하였다. OKL4는 유일하게 동기식 IPC를 버리고 비동기식인 가상 IRQ를 도입하였다[8].

초기 L4에서는 고성능을 고려하여 통신 종착점(IPC destination)으로 고유 스레드 ID를 사용했다. 하지만 이 방식은 낮은 보안성과 스레드 ID를 알리기 위한 부가적인 통신 오버헤드가 필요했다. 최근에는 EROS[13]로부터 영향을 받은 capability 메커니즘을 적용한 IPC endpoint가 seL4와 Fiasco.OC에서 통신 종착점으로 사용되고 있다.

IPC 메시지 형식에는 짧은 메시지와 긴 메시지가 있다. 짧은 메시지는 보통 물리적 레지스터만 사용하여 복사 연산을 수행하므로 속도가 빠르지만 보낼 수 있는 메시지가 제한적이고 아키텍처 의존적이다. 반면 긴 메시지는 메모리도 사용해서 다양한 메시지 형식을 사용할 수 있지만 페이지 부재(page fault)가 발생할 수밖에 없어 비효율적이다. 최근에는 짧은 메시지의 단점을 보완하고자 캐시를 레지스터처럼 사용하는 가상 메시지 레지스터를 도입하고, 거의 사용되지 않는 긴 메시지는 삭제하는 추세이다[8]. 현재 L4-embedded, NOVA, Fiasco.OC, seL4에서는 긴 메시지를 사용하지 않는다.

초기 L4는 긴 메시지를 이용하여 페이지 부재를 유도하는 DOS(Denial-of-Service) 공격을 당할 수 있어서 IPC 타임아웃을 두었다. 처음에는 이것이 괜찮은 방법이라 여겨졌지만, 타임아웃 값을 표현하기 위해 64비트 부동소수점을 사용해야 해서 복잡해지고 비용도 컸다. 최근 L4-embedded와 seL4에서는 IPC 타임아웃을 버리고, 폴링(0 타임아웃) 이나 블록(무한 타임아웃)로 설정할 수 있는 플래그 한 개로 대체되었다. OKL4는 비동기식 IPC만 사용

하므로 타임아웃 메커니즘을 사용하지 않는다[8].

Ⅲ. 스레드

다중 스레드 환경에서 각 스레드의 정보를 저장하고 있는 커널 객체가 TCB(Thread Control Block)이다. 커널이 스케줄링을 할 때 TCB 접근속도가 빨라야 비용을 줄일 수 있다. L4는 스레드 구분자(TID)를 인덱스로 하는 배열을 두고 찾고자 하는 스레드를 빠르게 접근할 수 있도록 했다.

CPU 하나에서 다중 스레드를 동시에 실행하기 위해서는 시간 다중화(time-multiplexing)가 필요하다. 이 때문에 적절한 스레드 스케줄링 방법이 요구되는데, L4 커널은 대부분 우선순위 기반 라운드로빈(round-robin) 방식을 사용했다. 또 초기 L4에서는 고비용인 큐 삽입과 삭제 동작이 빈번하게 일어나는 것을 막기 위하여 2장에서 언급한 lazy 스케줄링을 사용했지만, 최근 seL4는 최악의 경우에도 일정한 성능을 보장하는 Benno 스케줄링을 사용하고 있다.

2장에서 언급한 직접 스레드 전환의 사용 방식은 최근 L4 버전에서 실시간성을 감안하여 해당 스레드의 우선순위에 문제가 없으면 직접 스레드 전환을 사용하고 그렇지 않으면 스케줄러를 부르는 방식으로 바뀌었다.

Ⅳ. 예 외

커널 코드가 실행 중에 예외(exception)가 발생하면 그것을 빨리 처리하게 하는 것이 응답성과 실시간성 확보에 도움이 된다. 하지만 이런 접근 방식은 동시성 제어(concurrency control)가 필요하므로 커널 구현을 더 복잡하게 만든다.

전통적인 L4는 일반적으로 커널 동작 중 인터럽트가 발생되지 않게 하여 비선점형으로 실행되고, 장기 실행 작업(long running operations) 시 몇몇 지점에서만 인터럽트가 발생하게 하여 선점형으로 실행하게 만들었다. 초기 L4 ABI는 장기 실행되는 시스템 호출을 많이 가지고 있었기 때문에, 이 ABI을 이용한 Fiasco는 실시간 성능을 높이기 위해 아예 선점형으로 구현되었다. 하지만 다음 버전의 ABI에서 장기 실행되는 시스템 호출이 제거되면서 다음 버전인 Fiasco.OC에서도 다시 비선점형으로 회귀했다. 결국 최근 seL4에서는 비선점형을 선택하면서도 실시간성을 높이기 위해 주어진 인터럽트 대기시간(latency)이 제한선을 넘지 않게 구현하는 방향으로 가고 있다.

Ⅴ. 메모리

초기 L4에서 각 프로세스의 주소 공간 관리는 각 프로세스가 수행하는 map, grant, unmap 동작을 사용한 재귀적 페이지 매핑을 제공했다[9]. Liedtke가 고안한 이 방법은 간단하고 좋은 방법처럼 보였으나 그 뒤 여러 응용에서 사용되면서 비효율적인 커널 메모리 소모 문제와 보안 취약점이 발견되었다[8]. L4-embedded는 하드웨어를 보다 가깝게 흉내내어 모든 매핑이 항상 물리적 메모리 프레임의 범위에서 수행되도록 수정했고, OKL4 역시 L4-embedded의 방식을 따르면서 조금 수정했다. seL4는 take and grant 모델[14]을 사용하는 capability 방식을 사용하여 메모리 제어를 수행했다. 그 밖에 NOVA와 Fiasco.OC는 재귀적 매핑 방식을 그대로 사용하면서 제한을 추가하는 방식을 도입했다.

커널 힙의 공유 유무는 항상 논쟁 꺼리였다. Liedtke는 프로세스별 커널 힙 방식을 제안했고 NOVA, Fiasco, OKL4는 이 방식을 따른다. 반면에 seL4는 커널 힙을 전혀 쓰지 않고 모든 커널 객체를 구체적으로 정의하면서 capability 기반 접근 제어 방식을 사용한다[8].

Ⅵ. 기타

seL4는 C 언어로 구현하면서도 좋은 성능을 얻을 수 있는 fast-path 코딩 방식[15]을 개발하여 ARM11에서 one-way IPC에 188 사이클을 얻었다. 이런 연구결과는 어셈블리로 구현한 커널이 더 이상 성능에 관하여 절대적으로 유리하지 않음을 보여준다. 수작업으로 fast-path를 이용하면 파이프라인 멈춤(stall)을 피할 수 있고, 인라인 어셈블리를 사용하여 해당 아키텍처에 최적화시킬 수 있다. 처음 고급 언어로 작성된 L4는 Fiasco인데 C++을 사용했고, Pistachio도 C++를 사용했다. 이처럼 L4 개발에 C++를 많이 사용하였지만 C++ 사용의 장점은 잘 보이지 않는데, 특히 임베디드 분야에서는 좋은 C++ 컴파일러가 많지 않다. seL4의 경우는 verification 문제가 있어 C 언어를 사용하였다[8].

Ⅶ. 결론

본 연구에서는 L4 계열 마이크로커널이 사용한 성능 향상 기술들을 분석하고 이들 기술의 변천 과정을 살펴보았다. 요약하면 IPC에서 비동기식 IPC와 가상 메시지 레지스터 개념이 도입되고, 긴 메시지와 IPC 타임아웃 등은 제거되었다. 스레드 부분에서는 TCB 구조체를 배열로 사용하고 스레드 스케줄링은 우선순위 기반의 라운드 로빈 방식이 주로 쓰이는데 Liedtke가 제안한 lazy 스케줄링은 최악의 경우에 대한 성능이 일정하지 않아서 Benno 스케줄링 방식이 seL4에서 쓰이고 있다. 또 예외처리 부분에서는 대부분의 커널 코드가 짧은 실행시간을 가지므로 비선점형으로 구현하고 장기 실행시간을 가질 경우에는 몇몇 지점에 대해서만 부분적으로 선점형으로 실행되게 수정되었다. 메모리 관리에서는 재귀적 페이지 매핑 기술의 비효율성이 드러나 물리적 프레임을 사용하는 capability 방식으로 바뀌었고, 커널 힙은 프로세스 별로 두거나 전혀 사용하지 않는다. 마지막으로 어셈블리만으로 구현한 초기 L4와는 달리 최근 버전들은 약간의 성능 저하를 감수하고 C 언어와 같은 고급언어의 사용을 늘리는 추세이다. 여기서 살펴본 기술들은 매니코어 시스템용 새로운 마이크로커널을 설계 개발하는데 활용할 예정이다.

Acknowledgement: 본 연구에서 L4 기술의 분석을 위하여 L4를 설계 개발한 Liedtke의 논문 [9], [12] 및 그 동안 L4의 개선 발전에 기여한 Heiser의 논문 [8]의 내용을 많이 인용 활용하였습니다.

참 고 문 헌

[1] Multi-core processor, http://en.wikipedia.org/wiki/Multi-core_processor

[2] A. Vajda. “Programming Many-Core Chips”. Springer Science+Business Media, 2011.

[3] HeliOS, http://en.wikipedia.org/wiki/HeliOS

[4] S. Boyd-Wickizer, H. Chen and, R. Chen. “Corey: An Operating System for Many Cores”. In Proceedings of the 8th Symposium on Operating Systems Design and Implementation, San Diego, CA, USA, December 2008.

[5] A. Baumann, P. Barham and P. E. Dagand. “The Multikernel: A New OS Architecture for Scalable Multicore Systems”. In Proceedings of the 22nd ACM Symposium on OS Principles, Big Sky, MT, USA, October 2009.

[6] D. Wentzlaff, C. Gruenwald III and N. Beckmann. “A Unified Operating System for Clouds and Manycore: FOS”. 1st Workshop on Computer Architecture and Operating System co-design (CAOS), January 2010.

[7] Tessellation OS, http://tessellation.cs.berkeley.edu

[8] K. Elphinstone, and G. Heiser. “From L3 to seL4: What Have We Learnt in 20 Years of L4 Microkernels?”. 24th ACM SIGOPS Symposium on Operating Systems Principles, Farmington, PA, USA, pages 133 - 150, 2013.

[9] J. Liedtke. “On μ-Kernel Construction”. In Proceedings of the 15th ACM Symposium on Operating Systems Principles, pages 237-250, Copper Mountain, CO, USA, December 1995.

[10] Mach, http://en.wikipedia.org/wiki/Mach_(kernel)

[11] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and Jean Wolter. “The Performance of μ-kernel-based Systems”. Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP), pages 66-77, St. Malo, France, October 1997.

[12] J. Liedtke. “Improving IPC by Kernel Design”. In Proceedings of the 14th ACM Symposium on Operating Systems Principles, pages 175-188, Asheville, NC, USA, December 1993.

[13] J. S. Shapiro, J. M. Smith, and D. J. Farber. “EROS: A Fast Capability System”, In Proceedings of the 17th ACM Symposium on Operating Systems Principles, pages 170-185, Charleston, SC, USA, December 1999.

[14] R. J. Lipton and L. Snyder. “A Linear Time Algorithm for Deciding Subject Security”. Journal of the ACM, 24(3):455-464, 1977.

[15] B. Blackham, Y. Shi, and G. Heiser. “Improving interrupt response time in a verifiable protected microkernel”. In Proceedings of the 7th EuroSys Conference, pages 323-336, Bern, Switzerland, April 2012.