ETRI

2018
Electronics and
Telecommunications
Trends

유니커널의 동향과 매니코어 시스템에 적용

Trends in Unikernel and Its Application to Manycore Systems

차승준 (S.J. Cha, seungjunn@etri.re.kr) 차세대 OS 기초연구센터 선임연구원
전승협 (S.H. Jeon, shjeon00@etri.re.kr) 차세대 OS 기초연구센터 선임연구원
람 닉 (Ramneek, ramneek@etri.re.kr) 차세대 OS 기초연구센터 박사후연구원
김진미 (J.M. Kim, jinmee@etri.re.kr) 차세대 OS 기초연구센터 책임연구원
정연정 (Y.J. Jeong, yjjeong@etri.re.kr) 차세대 OS 기초연구센터 책임연구원
정성인 (S.I. Jung, sijung@etri.re.kr) 차세대 OS 기초연구센터 책임연구원/센터장

As recent applications are requiring more CPUs for their performance, manycore systems have evolved. Since existing operating systems do not provide performance scalability in manycore systems, Azalea, a multi-kernel based system, has been developed for supporting performance scalability. Unikernel is a new operating system technology starting with the concept of a library OS. Applying unikernel to Azalea enables an improvement in performance. In this paper, we first analyze the current technology trends of unikernel, and then discuss the applications and effects of unikernel to Azalea. Azalea-unikernel was built in a single image consisting of libOS, runtime libraries, and an application, and executed with the desired number of cores and memory size in bare-metal. In particular, it supports source and binary compatibility such that existing linux binaries can be rebuilt and executed in Azalea-unikernel , and already built binaries can be run immediately without modification with a better performance. It not only achieves a performance enhancement, it is also a more secure OS for manycore systems.

* DOI: 10.22648/ETRI.2018.J.330613

* 이 논문은 2018년도 정부(과학기술정통부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임[No.2014-3-00035, 매니코어 기반 초고성능 스케일러블 OS 기초연구센터(차세대OS기초연구센터)].

Ⅰ. 서론

고성능 컴퓨팅, 빅데이터 처리, 인메모리 데이터베이스 관련된 응용프로그램이 발전으로 처리해야 하는 데이터가 많아지고 이에 따라 빠른 처리속도를 제공하는 시스템이 필요하게 되었다. 이에 따라 인텔, AMD 등과 같은 주요 CPU 벤더뿐만 아니라 Tilera, Kalray 등에서도 수십에서 수백 코어의 CPU를 생산하고 이를 기반으로 인텔, 레노버, 슈퍼마이크로 시스템즈에서는 수백 코어 이상의 시스템을 생산하고 있다. 특히 수백에서 수천 개의 프로세스 코어로 구성된 시스템을 매니코어 시스템이라 한다[1].

매니코어 시스템에서 범용모노리틱 운영체제인 리눅스는 아직까지 코어 수 증가에 따른 확장성을 지원하지 못한다[1]. 한국전자통신연구원에서는 매니코어 시스템에서 성능 확장성 지원을 위해 코어의 성능과 역할에 따라 커널의 기능을 분산하여 구성하는 멀티커널 기반 운영체제인 Azalea를 개발하였다[2].

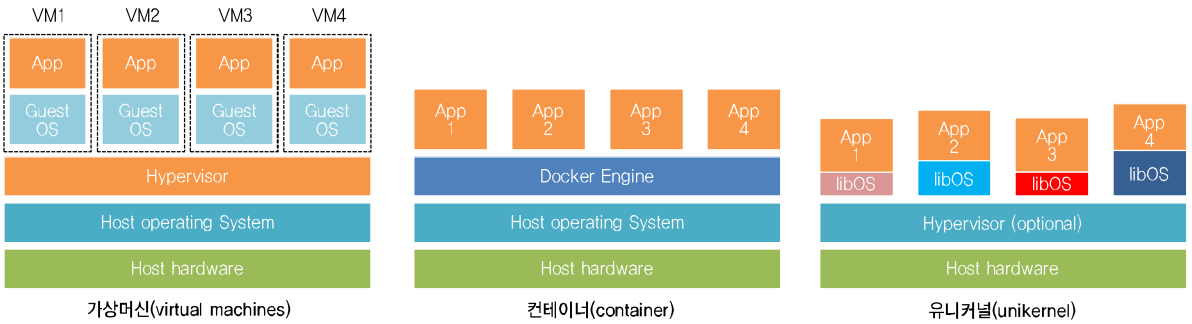
유니커널이란 클라우드 컴퓨팅에서 동작하는 새로운 운영체제 기술로 라이브러리 운영체제의 개념으로부터 시작하였다. 주요 특징으로는 응용프로그램이 필요한 라이브러리만을 가지고 있고 응용프로그램과 커널이 같은 주소공간을 사용한다는 것이다. 따라서 실행 이미지의 크기가 작고 특정 응용프로그램에 최적의 성능을 제공하며 부팅이 빠르고 불필요한 커널의 기능이 없기 때문에 공격 대상이 작아서 보안에도 강하다는 장점을 가진다.

Azalea에서는 경량커널을 통해 커널의 간섭을 줄여 성능과 확장성을 확보하였지만, 여전히 문맥교환이나 시스템콜이 많이 발생하는 응용프로그램의 경우 성능에 제약이 발생한다. 이러한 Azalea의 경량커널에 유니커널의 기법을 적용하면 문맥교환이나 시스템콜 처리시 비용이 감소하여 성능이 향상된다.

이에 본고에서는 현재 유니커널의 기술 동향에 대해 살펴보고, 매니코어 시스템의 성능 확장성 지원 및 성능 향상을 위해 Azalea 멀티커널에 유니커널의 적용 및 그 효과에 대해 설명한다.

Ⅱ. 유니커널

유니커널 관련 프로젝트를 관리하는 사이트인 유니커널 시스템즈(Unikernel.org)에서 유니커널을 “유니커널은 특정 응용프로그램에 특화된, 단일 주소공간을 사용하고 라이브러리 운영체제의 방식으로 생성한 시스템 이미지”로 정의하였다. [3], [4] 또한 [5]에서는 “유니커널은 고급 언어로 작성되고 개별 소프트웨어 구성요소처럼 동작하는 특화된 운영체제이다. 전체 응용프로그램(또는 어플라이언스)은 분산시스템 환경과 비슷하게



(그림 1) 가상머신, 컨테이너, 유니커널 비교

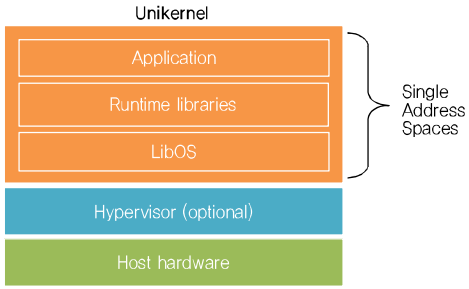
[출처] Reproduced form H. Mahkonen, “Unikernels meet NFV,” Ericsson Research Blog, June 2, 2016, https://www.ericsson.com/research-blog/unikernels-meet-nfv/

여러 개의 유니커널로 구성된다"고 정의하고 있다. 유니커널은 클라우드 환경에서 많이 사용되어 왔지만, 요즘에는 IoT(Internet of Things)나 베어메탈 환경에서도 많이 사용하고 있다. 현재 도커는 유니커널 시스템즈를 인수하여 도커 기반 유니커널에 대한 연구를 진행하고 있다.

가상머신, 컨테이너, 유니커널의 차이는 (그림 1)과 같다. 유니커널은 가상머신과 컨테이터에서 공유하는 운영체제나 엔진을 포함하고 있지 않다.

유니커널은 IoT 또는 클라우드 시스템을 위한 라이브러리 운영체제 개념으로부터 시작된 새로운 운영체제 기술이다. 유니커널은 독립적으로 실행할 수 있는 단일 부팅 이미지로 만들어진다. 또한, 하이퍼바이저 위에서 동작하기 때문에 클라우드 환경에서 사용되지만, 또한 베어메탈에서 동작이 가능하다. 클라우드 환경에서 사용하는 경우 외부 종속성을 가지지 않기 때문에 모든 하이퍼바이저와 플랫폼에서도 거의 동일하게 실행된다.

유니커널의 구조는 (그림 2)와 같다. 유니커널의 구성 요소로는 커널 기능을 담당하는 libOS, 응용프로그램 실행에 필요한 런타임 라이브러리, 응용프로그램으로 구분된다. 필요한 라이브러리들은 하나의 목적을 가진 프로그램으로 링크되어 최소한의 부팅 가능한 이미지로 만들어진다. 사용되지 않은 코드는 링크시에 제거되고 최소한의 필요한 디바이스 드라이버들만 포함된다. 예



(그림 2) 유니커널의 구조

[출처] Reproduced form H. Mahkonen, "Unikernels meet NFV," Ericsson Research Blog, June 2, 2016, https://www.ericsson.com/research-blog/unikernels-meet-nfv/

를 들어, 웹서버의 경우 최소한의 요구사항으로 설치되는 우분투는 5GB의 용량이 필요한 반면, 이미지나 웹컨텐츠를 포함하지 않은 유니커널 기반 웹서비스의 경우 2MB보다 작은 이미지로 만들어진다. 이미지의 크기가 줄어듦에 따라서 부팅시간이 빨라지는 장점을 가진다.

유니커널이 부각된 이유는 현재 가장 많이 사용되는 모노리틱 운영체제인 리눅스가 점점 발전하면서 점점 용량이 커지고 무거워지기 때문이다. 현재 리눅스 배포판에는 수억 라인의 코드로 구성되어 있지만, 일반적인 응용프로그램은 필요한 경우, 필요한 부분의 코드만을 사용기 때문에 대부분의 코드는 사용되지 않는다. 최근 배포된 커널의 코드는이미 140,000 LOC(Line Of Code) 이상으로 구성되어 있으며, 배포판에는 기본적으로 수백만 개의 사전 설치된 프로그램과 라이브러리를 포함하고 있다. 리눅스는 어느 하드웨어에서나 동작할 수 있도록 설계되었기 때문에 플로피 디스크 드라이버와 같이 불필요한 기능들을 포함하고 있다.

유니커널은 응용프로그램이 필요한 라이브러리만을 가지고 있기 때문에 공격 대상이 크게 감소하여 보안에 강한 장점을 가진다. 심지어는 범용 운영체제에서 필수적인 셸이나 터미널을 제공하지 않기 때문에, 시스템의 상태에 대한 확인이 필요할 경우 유니커널에서 제공하는 API를 사용하여 요청하고 읽기 전용의 결과만 확인 가능하다. 가상머신에서 유니커널을 실행할 때 하이퍼바이저에서 제공되는 격리 정책을 통해 추가적인 보안 효과를 얻을 수 있다.

유니커널의 큰 특징 중 하나는 단일 주소공간을 사용한다는 것이다. 단일 주소공간을 사용하면 커널모드와 유저모드간에 같은 권한레벨을 사용하지 때문에 인터럽트 등 발생시 문맥교환이 발생하지 않으며, 시스템콜 또한 함수 호출 방식을 사용하여 문맥교환이 발생하지 않는다. 문맥교환이 발생하지 않으면 기존 커널에서 주소공간 보호를 위해 사용된 MMU 강제 격리(MMU-enforced isolation)가 불필요하고, 레지스트리를 저장/

복원하는 비용을 줄일 수 있기 때문에 성능을 향상된다. 문맥교환시 권한이 하이퍼바이저로 트랩되는 가상화 환경에서도 성능이 향상된다. 시스템콜의 경우 베어 메탈 환경에서 2번의 문맥교환이 발생하지만, 가상화 환경에서는 4번의 문맥교환이 필요하기 대문이다. 가상화 환경에서는 문맥교환을 완료하는데 수 마이크로초가 걸릴 수 있으며 특히 캐쉬 메모리가 오염된 경우 추가적인 비용이 발생할 수 있다[6], [7].

이미지 사이즈가 작을 뿐만 아니라 불필요한 라이브러리들이 없기 때문에, 유니커널의 부팅과 종료시간은 수 밀리초 단위로 매우 짧다. 특히 '즉시 실행(just in time)'을 중요시하는 클라우드 환경에서 서비스가 필요할 때 빠르게 생성되고 끝나면 바로 종료되어야 하는데 유니커널은 이러한 특성에 맞게 실행될 수 있다.

유니커널의 단점은 다음과 같다. 현재 유니커널은 범용적인 서비스가 아닌 웹서버와 같은 마이크로서비스에 적합하기 때문에 일반적인 사용성에 제약에 있다. 또한, 라이브러리 운영체제가 먼저 설정되어야 하기 때문에 다른 서비스가 필요한 경우 완전히 다른 패키지를 제공하여야 하는 등 유연성에 제약이 있다. 마지막으로 유니커널을 디버깅과 같은 개발 환경을 충분히 제공하고 있지 않기 때문에 기존 운영체제보다 응용프로그램을 개발하기 어렵다.

Ⅲ. 유니커널 연구 동향

유니커널 관련 기존 연구들에서는 유니커널을 크게 개발유형에 따라 클린 슬레이트(Clean-slate) 타입과 레거시(Legacy) 타입으로 구분한다[8]. 각 타입별 대표적인 프로젝트는 〈표 1〉과 같다.

〈표 1〉 유니커널의 대표적인 프로젝트

클린 슬레이트	레거시
MirageOS(Ocaml)	OSv
HaLVM(Haskell)	Rumprun
LING(Erlang)	HermitCore
IncludeOS(C/C++)	ClickOS
	DrawBridge

클린 스테이트 타입의 유니커널은 성능 최적화를 위해 특정한 프로그래밍 언어로 개발되며, 스크래치부터 새롭게 작성한다. 대상 응용프로그램이 필요한 운영체제 기능들로만 구성하였기 때문에 코드 크기가 매우 작고 프로그램 언어의 특징을 반영할 수 있다는 장점을 가진다. 반면 응용프로그램을 작성하는데 있어서는 구현 언어 및 기능에 대해 잘 알아야 될 뿐만 아니라, 응용프로그램 개발 시 커널에서 제공하지 않는 기능을 따로 추가하여야 하는 등 어려움이 있다.

레거시 타입 유니커널은 POSIX 표준 시스템콜을 지원한다는 것이 특징이다. 이를 통해 POSIX 기반으로 개발된 리눅스 응용프로그램을 변경없이 유니커널에서 실행할 수 있다. 뿐만 아니라 기존에 개발된 런타임 시스템 라이브러리를 사용할 수 있지만, 그만큼 코드 크기가 커질 수 있다. 하지만, 리눅스에서와 같은 방식을 개발할 수 있기 때문에 개발자는 쉽게 응용프로그램을 개발할 수 있다.

1. 클린 슬레이트 타입

가. MirageOS

MirageOS[4]는 다양한 클라우드 컴퓨팅 및 모바일 플랫폼에서 안전한 고성능 네트워크 응용프로그램을 위한 유니커널로 리눅스 파운데이션의 인큐베이터 프로젝트 중 하나이다. MirageOS 프로젝트는 초기 유니커널의 개념을 정립하고 퍼트리는데 크게 영향을 주었다. 이후 많은 개발자가 유니커널 시스템즈(Unikernel Systems)를 설립하기 위해 이직하였다.

MirageOS를 통해 Xen 하이퍼바이저에 배포할 수 있는 유니커널을 생성할 수 있다. MirageOS는 2014년도에 2.0 버전을 배포하였고, 현재 안정적으로 사용 가능하다. 프로젝트 웹 사이트에는 초보자가 유니커널을 구

축하는 데 도움이 되는 문서와 테스트 코드를 제공한다. 현재 100개 이상의 MirageOS 라이브러리가 있으며, OCaml 생태계 내에서 호환 라이브러리가 점점 더 늘어나고 있다.

나. HaLVM

HaLVM(Haskell Lightweight Virtual Machine)은 갈루아(Galois Inc.)에서 지원하는 오픈소스 프로젝트 중 하나로, 하스켈 프로그래밍 언어로 작성되는 유니커널이다. 개발자는 HaLVM이 포함된 Glasglow Haskell 컴파일러 도구를 사용하여 Xen 하이퍼바이저에서 직접 실행할 수 있는 유니커널을 만들 수 있다[9]. HaLVM의 초기목적은 운영체제의 핵심 구성요소의 설계 및 테스트를 쉽게 하는 것이었지만, 현재에는 유니커널 개발을 위한 범용 플랫폼으로 발전하였다. 2.1 버전에서는 Xen 하이퍼바이저를 지원하여 서버나 데스크톱의 배포뿐만 아니라 아마존 EC2도 지원한다. 3.0 버전에서는 KVM이나 베어메탈의 환경을 지원할 수 있도록 개발 중이다.

다. LING

LING[3]은 Xen 커뮤니티에서 지원하는 프로젝트 중 하나인 Erlang on Xen에서 연구 중인 프로젝트로, Erlang 프로그래밍 언어를 사용하고 Xen 하이퍼바이저에서 동작하는 유니커널이다. 개발자는 Erlang을 사용하여 프로그램 코드를 생성하고 LING 유니커널로 배포할 수 있다. LING의 특징은 대다수의 벡터 파일을 제거하고 세 개의 외부 라이브러리만 사용한다는 것이다. 프로젝트 웹사이트도 유니커널 기반으로 서비스되고 있으며, 18MB 메모리만으로 웹서버가 실행되고 있음을 확인할 수 있다.

라. IncludeOS

IncludeOS는 하나의 목적을 위한 운영체제의 연구를 위해 2016년 설립한 IncludeOS에서 개발한, 클라우드 서비스를 위한 최소한의 기능만을 제공하는 라이브러리 운영체제이다[10]. 특히, 가상 하드웨어 환경에서 C++ 코드를 실행할 수 있는 환경을 제공한다. IncludeOS은 빌드 시스템을 통해 커널과 표준 라이브러리, C++응용 프로그램이 통합 빌드되어 실행 이미지가 만들어진다. 현재 생성된 이미지는 주로 KVM 하이퍼바이저에서 테스트를 진행하고 있으며, VirtualBox와 Boch에서도 실행 가능하다.

2. 레거시 타입

가. OSv

OSv는 Cloudius Systems에서 클라우드 가상머신을 위해 개발한 새로운 형식의 운영체제이다[11]. 거의 모든 응용프로그램을 유니커널로 바꾸어 동작시키고자 하였으며, 이와 같이 범용 유니커널의 제공은 다른 프로젝트와 차별성을 가진다. 하지만 KB 단위의 이미지를 제공하는 다른 프로젝트들과는 달리 MB 단위의 이미지를 제공한다. OSv는 KVM, Xen, Virtualbox, VMware의 하이퍼바이저 위에 실행하도록 설계되어, 1초이내 부팅을 지원하는 등 탁월한 성능과 신속하고 간편한 관리를 지원한다. 뿐만 아니라, C, C++, JVM, Ruby 및 Node.js 소프트웨어 스택을 지원한다. Cloudius Systems는 ScyllaDB로 합병되어 더 이상 OSv에 대한 서비스를 지원하지 않지만, 현재 OSv 오픈소스 커뮤니티에서 소프트웨어를 관리한다.

나. Rumprun

Rumprun은 NetBSD rump 커널을 기반으로 연구 중인 운영체제로, POSIX를 지원하는 소프트웨어를 변경 없이 유니커널로 실행할 수 있게 지원한다[12]. 파일시스템, POSIX 시스템 콜 핸들러, PCI 디바이스 드라이버, SCSI 프로토콜 스택, virtio, TCP/IP와 같은 커널 드

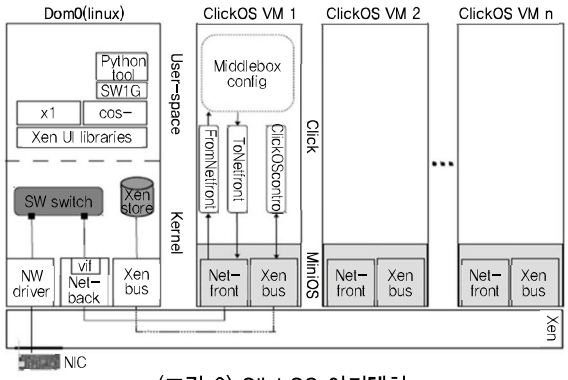
라이버를 제공한다. 이론적으로는 리눅스나 유닉스의 응용프로그램을 단일 유니커널로 컴파일하여 사용이 가능하다. Rumprun은 ARM 하드웨어부터 Xen과 같은 하이퍼바이저까지 다양한 하드웨어 플랫폼을 지원한다.

다. Hermitcore

HermitCore는 독일의 아헨(Aachen) 공과대학교에서 연구 중인 프로젝트로 HPC와 클라우드 컴퓨팅을 위해 개발한 유니커널이다[13]. 이는, 멀티커널에 유니커널의 기능을 확장한 구조를 제안하여 성능 확장성을 제공한다. 생성된 이미지는 KVM 하이퍼바이저에서 직접 실행할 수 있지만 기본적으로 인텔 64비트 아키텍처의 베어메탈에서도 실행할 수 있다. Hermitcore의 소프트웨어 스택은 하드웨어 위에 libOS가 커널을 담당하여 Newlib과 OpenMP/MPI의 런타임 라이브러리로 구성된다. 리눅스에 설치된 Proxy를 통해 유니커널이 로딩되며, C 라이브러리 지원을 위해 Newlib을 사용한다. 또한 현재버전은 C/C++, 포트란, Go, Pthread, OpenMP와 메시지 전달을 위한 iRCCE를 지원한다.

라. ClickOS

ClickOS는 유니커널을 이용하여 네트워크 기능 가상화(NFV)를 지원한다[(그림 3) 참조], [15]. ClickOS는 유럽 NEC의 시스템과 머신러닝(Systems and Maching Learning) 실험실에서 연구하는 프로젝트 중 하나이다. ClickOS 로 생성된 유니커널 기반 가상머신은 이미지 크기는 5MB 정도이며, 20ms 정도로 빠르게 부팅되고, 45ms의 낮은 지연시간을 제공한다. 또한 10GB 네트워크를 모두 사용하면서 100개 이상을 동시에 실행할 수 있다. ClickOS는 Xen 하이퍼바이저에 쉽게 배포할 수 있다는 장점을 가진다.



(그림 3) ClickOS 아키텍처

[출처] Reprinted from J. Martins et al., "ClickOS and the Art of Network Function Virtualization," *USENIX Symp. Netw. Syst. Des. Implement. (NSDI□14)*, Seattle, WA, USA, Apr. 2-4, 2014, pp. 259-473.

마. DrawBridge

Drawbridge는 마이크로소프트 리서치에서 진행하는 프로젝트로, 윈도우용 응용프로그램 샌드박싱을 위한 새로운 가상화를 지원한다[16]. Drawbridge는 피코프로세스(picopress)와 라이브러리 운영체제로 구성되며, 프로세스 기반으로 격리된 컨테이너로 최소한의 커널 API만을 통해 접근이 가능하다.라이브러리 운영체제는 윈도우 운영체제를 수정하여 응용프로그램이 피코프로세스에서 효과적으로 실행될 수 있도록 하는 기능을 제공한다. Drawbridge는 피코프로세스와 라이브러리 운영체제를 사용하여 자원 사용을 줄이면서, 보안에 강하고, 영속적인 호환성 및 실행 연속성을 제공한다.

Ⅳ. 매니코어 시스템과 멀티커널, 유니커널

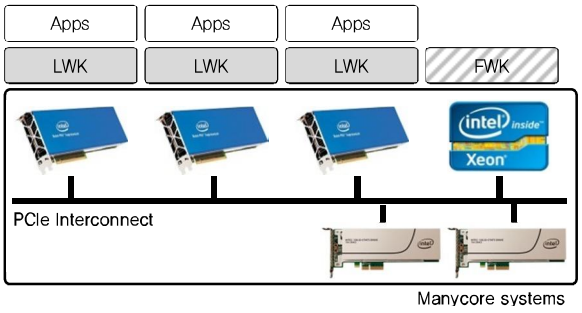
가. Azalea: 매니코어 시스템용 멀티커널

제조 공정이 발전함에 따라 CPU는 싱글코어에서 여러개의 코어를 칩에 집적하는 멀티코어로 발전하였다. 최근 응용프로그램이 엑사스케일 컴퓨팅과 같이 빠른 처리를 요구하게 되자 많은 CPU 벤더들은 멀티코어를 지나 매니코어 시스템을 개발하고 있다.

매니코어 시스템이 발전함에 따라서, 코어 증가에 따른 성능 확장성 지원이 필요하였다. 멀티커널은 코어의

성능과 역할에 따라 커널의 기능을 분산하는 구조로 구성한다[17]. 본 연구진에서는 매니코어 시스템용 멀티 커널 운영체제인 Azalea를 개발하였다.

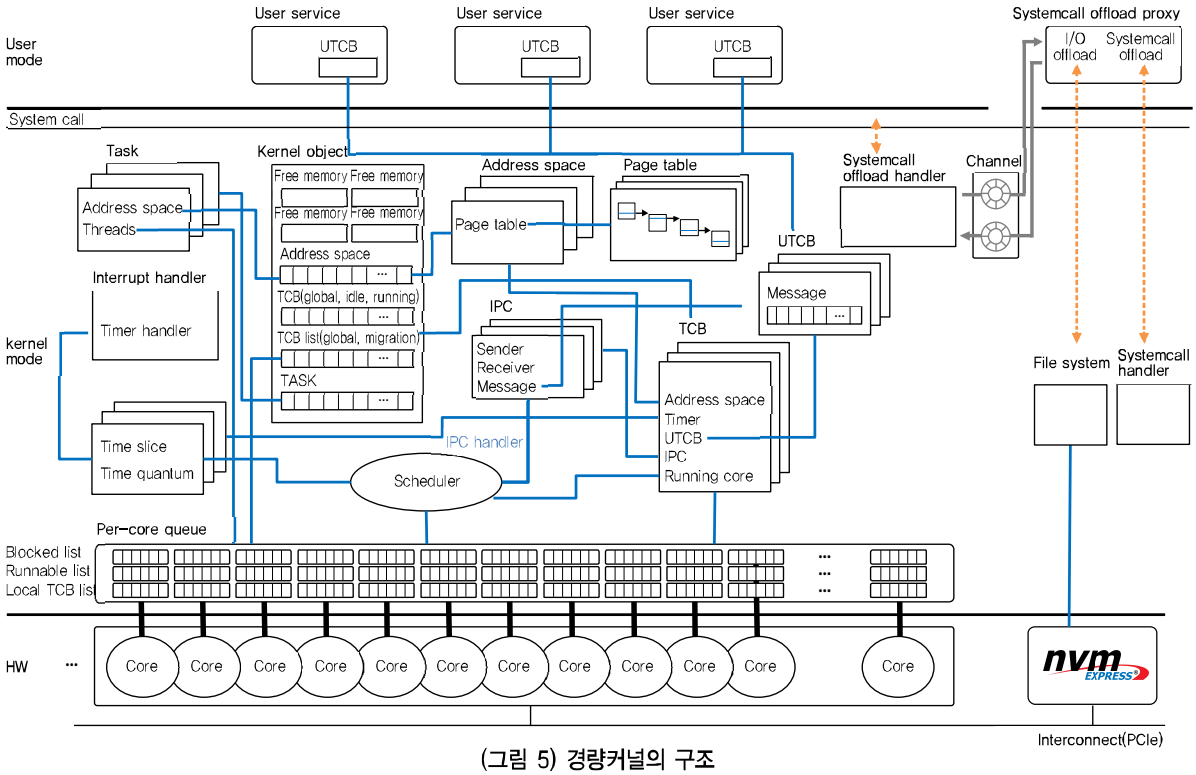
Azalea는 코어의 성능과 역할을 고려하여, 풀커널(FWK: full weight kernel)과 경량커널(LWK: light-weight kernel)로 구분하여 커널을 분산한다[(그림 4 참고)]. 풀커널은 코어의 수는 적지만 성능이 빠른 코어에 설치하여 리눅스와 같은 풍부한 커널 기능을 제공한다. 경량커널은 코어의 성능은 낮지만 많은 코어에 설치하고 응용프로그램 수행에 필요한 최소한의 커널 기능을 제공한다. 경량커널은 파일 및 네트워크 입출력과 같은 소프트웨어 스택을 제공하지 않고 풀커널에 오프로드(offload)하여 처리할 수 있는 매커니즘을 제공하여 커널의 기능을 최소화한다.



(그림 4) Azalea의 구조

[출처] Reprinted from S.-J. Cha et al., "Multi-Kernel based Scalable Operating Sys-tem for Manycore Systems," *Future Generation Inform. Technol.*, vol. 148, 2017, pp. 28-34

경량커널의 구조는 (그림 5)와 같다. 경량커널은 성능 확장성 확보를 위해 커널이 기능을 단순화하였고, 공간 분할 기법을 통해 커널에서 사용하는 공유자원을 줄였다. 기본적으로 태스크, 주소관리, 스케줄링의 기능을 제공하며 저수준 하드웨어 추상화를 지원한다. 확장성 지원을 위한 코어별 자료구조를 지원하며 락(lock) 등의



(그림 5) 경량커널의 구조

[출처] Reprinted from S.-J. Cha et al., "Multi-Kernel based Scalable Operating Sys-tem for Manycore Systems," *Future Generation Inform. Technol.*, vol. 148, 2017, pp. 28-34

자원 경쟁을 최소화하였다.

이러한 멀티커널 구조로 이기종 매니코어 시스템에서 기존 모노리틱 커널 구조보다 성능 확장성을 제공할 수 있음을 확인하였다[2].

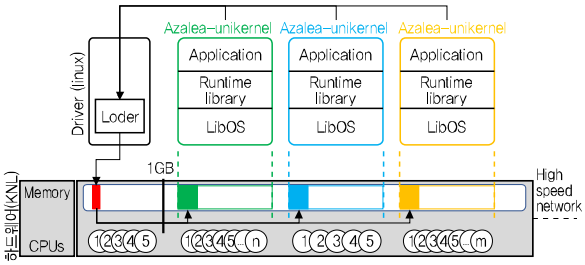
나. 유니커널의 적용

Azalea는 매니코어 시스템에서 멀티커널을 구성하여 성능 확장성을 지원한다. 이를 위해 응용프로그램이 실행되는 코어에 경량커널을 개발하여 커널의 기능을 최소화하고, 커널에서 발생하는 간섭을 최소화하여 커널이 응용프로그램만을 실행할 수 있는 환경을 제공하였다.

하지만 경량커널에서는 하나의 커널을 응용프로그램들이 공유해서 사용하게 때문에 커널 자료구조에 대한 자원 경쟁이 여전히 남아있다. 또한, 사용자 공간과 커널 공간이 분리되어 있어서 문맥 교환시 데이터를 전달하고 컨텍스트 유지를 위해 레지스트리 값을 저장/복원하는데 비용이 발생한다. 이러한 추가적인 비용은 전체적인 시스템 성능을 저해한다.

유니커널은 앞에서 살펴본 것과 같이, 실행에 필요한 라이브러리로만 구성되며, 커널과 응용프로그램이 같은 주소 공간을 사용하기 때문에 문맥교환시 추가적인 비용이 발생하지 않는다. 본 연구에서는 이러한 특징을 Azalea의 경량커널에 적용하여 Azalea-unikernel을 개발하였다. Azalea-unikernl은 멀티커널에 유니커널의 장점이 반영되어, 응용프로그램이 실행에 필요한 커널 기능과 라이브러리로만 구성하 이미지 크기를 줄이고 빠른 부팅을 지원한다. 이를 통해 응용프로그램의 실행 성능은 더욱 향상된다.

Azalea-unikernl의 구조는 크게 커널기능을 담당하는 libOS와 런타임 라이브러리, 응용프로그램으로 구성된다[(그림 6) 참조]. 하나의 서버는 여러 개의 Azalea-unikernel을 각각 원하는 코어의 수와 메모리 크기로 실행할 수 있다. 서버의 일부에는 리눅스를 설치하여 각각



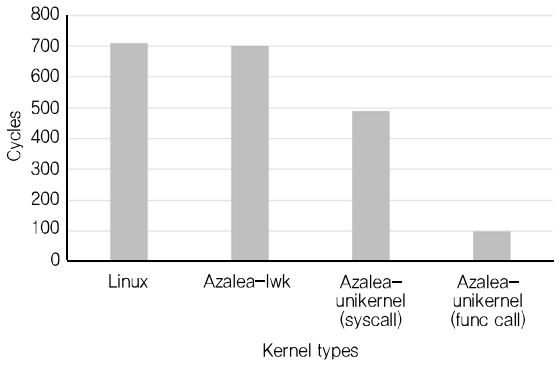
(그림 6) Azalea-unikernel의 구조

의 커널이 로딩하거나 다른 노드간 통신을 지원하는 드라이버 역할을 수행한다.

또한, Azalea-unikerel에서는 소스 호환성과 바이너리 호환성을 지원한다. 소스 호환성은 기존에 작성된 응용프로그램 소스코드를 다시 빌드하여 실행하는 환경을 의미한다. 일반적인 응용프로그램에서는 커널의 주소공간에 대한 보안성을 위해 시스템콜을 사용하여 커널의 함수를 수행하는데 반면, 유니커널에서는 응용프로그램과 커널이 하나의 주소공간을 사용하기 때문에 직접 함수의 호출이 가능하다. 시스템콜 처리 방식을 함수호출로 교체함으로써 시스템콜 발생시 성능이 향상된다. 바이너리 호환성은 기존의 소스코드를 바로 실행할 수 있는 환경이다. 이를 위해 기본적인 런타임 라이브러리(예. glibc, pthread 등)을 연결하여 실행 가능하도록 구현하였다. 이를 통해 기존에 빌드된 응용프로그램의 이미지를 바로 실행할 수 있다는 장점을 갖는다.

나. 성능평가

유니커널의 성능을 실험하기 위해서 'getpid()' 시스템콜을 10,000번 수행한 후 다른 커널과 실행 시간을 비교하였다. 'getpid()' 시스템콜은 현재 동작하고 있는 쓰레드의 ID 값을 읽어오는 가장 간단한 시스템콜이다. 성능 비교는 모노리틱 커널인 리눅스, 경량커널인 Azalea-lwk, 유니커널인 Azalea-unikernel을 대상으로 한다. 특히 Azalea-unikernel의 경우, 소스/바이너리 호환성에 따른 성능 비교를 위해 각각 나눠 실험을



(그림 7) Azalea-unikernel 시스템콜 실험 결과

진행하였다.

실험 결과는 (그림 7)과 같다. 리눅스와 Azalea-lwk의 경우 syscall 을 통해 시스템콜 호출이 유저 모드에서 커널 모드로 전달하고, sysret을 통해 다시 유저 모드로 돌아온다. 따라서 부가적인 문맥교환 오버헤드가 발생한다. Azalea-lwk의 경우 리눅스보다 문맥교환 등 커널 간섭이 적기 때문에 일부 성능이 좋음을 확인하였다. 바이너리 호환성을 제공하는 Azalea-unikernel은 마찬가지로 syscall과 sysret을 사용하지만 유니커널이기 때문에 문맥교환이 발생하지 않아 성능이 리눅스와 Azalea-lwk 보다 좋음을 확인할 수 있다. 소스호환성을 제공하는 Azalea-unikernel은 시스템콜을 syscall과 sysret을 통해 처리하지 않고 함수 호출로 처리하기 때문에 문맥교환뿐만 아니라 레지스트리를 저장하고 복원하는 절차가 필요하지 않다. 실험결과 리눅스 대비 최대 7.5배 성능이 향상된 것을 확인할 수 있다.

Ⅵ. 결론

유니커널은 클라우드 환경에서 응용프로그램이 작고, 안전하고, 빠르게 실행할 수 있는 기능을 제공한다. 국외 많은 대학 및 연구기관은 유니커널 기반의 연구들을 수행하여 이미 많은 유니커널이 개발하였으며, 유니커널 프로젝트를 관리하는 유니커널 시스템즈가 도커에 합병되어 발전하는 양상은 유니커널 성능이 증명되었다는 것을 보여준다.

매니코어 시스템이 점차 발전함에 따라서, 성능 확장성 문제는 반드시 해결되어야 할 문제이며 이를 해결하기 위해 멀티커널 기반 운영체제를 연구하였다. 멀티커널 운영체제에 유니커널의 기능들을 활용하면, 성능 측면에서도 이득이 있을 뿐만 아니라 나아가 더 보안에 강한 매니코어용 운영체제로 자리매김 할 수 있을 것이다.

약어 정리

API	Application Programming Interface
CPU	Central Processing Unit
GB	GigaByte
IoT	Internet Of Things
IP	Internet Protocol
JVM	Java Virtual Machine
KB	KiloByte
KVM	Kernel-based Virtual Machine
LOC	Line Of Code
MB	MegaByte
MMU	Memory Management Unit
NFV	Network Function Virtualization
POSIX	Portable Operating System Interface
TCP	Transmission Control Protocol

참고문헌

[1] 정성인 등, “매니코어 운영체제 연구현황 및 계획,” 전자통신동향분석, 제32권 제6호, 2017, pp. 83-95.

[2] S.-J. Cha et al., “Multi-Kernel based Scalable Operating System for Manycore Systems,” *Future Generation Inform. Technol.*, vol. 148, 2017, pp. 28-34.

[3] A. Madhavapeddy “Unikernels-Rethinking Cloud Infrastructure,” 2017, http://unikernel.org/

[4] Wikipedia, “Unikernels,” 2015, https://en.wikipedia.org/wiki/Unikernel

[5] A. Madhavapeddy and D.J. Scott, “Unikernels: Rise of the Virtual Library Operating System,” *Distrubuted Comput. Mag.*, vol. 11, no. 11, Nov. 2013, p. 30.

[6] B. Sigoure, “How Long Does It Take to Make a Context Switch?” 2010, https://blog.tsunanet.net/2010/11/how

-long-does-it-take-to-make-context.html

[7] C. Li, C. Ding, and K. Shen, "Quantifying the Cost of Context Switch," *Proc. Workshop Experimental Comput. Sci. (ExpCS '07)*, SanDiego, CA, USA, June 13-14, 2007, pp. 1-4.

[8] A. Madhavapeddy et al., "Unikernels: Library Operating Systems for the Cloud," *ACM SIGPLAN Notices*, vol. 48, no. 4, Apr. 2013, pp.461-472

[9] A. Wick et al., "Haskell Lightweight Virtual Machine (halvm)" GALOIS, INC., 2014. https://galois.com/project/halvm/

[10] A. Bratterud et al., "IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services," *IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Vancouver, Canada, Nov. 30-Dec. 3, 2015, pp. 250-257.

[11] A. Kivity et al., "OSv-Optimizing the Operating System for Virtual Machines," *USENIX Annu. Technical Conf.*, Philadelphia, PA, USA, June 19-20, 2014, pp. 61-72.

[12]A. Kantee, "On Rump Kernels and the Rumprun Unikernel," XenProject, 2015. https://blog.xenproject.org/

[13] Stafan Lankes et al., "A Low Noise Unikernel for Exrem-Scale Systems," *Architec. Comput. Syst.*, Vienna, Austria, Apr. 3-6, 2017, pp. 73-84.

[14] C.-C. Tsai et al., "Cooperation and Security Isolation of Library Oses for Multi-process Applications," *Eur. Conf. Comput. Syst. (EuroSys'14)*, Amsterdam, Netherlands, Apr. 14-16, 2014, pp. 9:1-9:14.

[15] J. Martins et al., "ClickOS and the Art of Network Function Virtualization," *USENIX Symp. Netw. Syst. Des. Implement. (NSDI'14)*, Seattle, WA, USA, Apr. 2-4, 2014, pp. 259-473.

[16] D.E. Porter et al., "Rethinking the Library OS from the Top Down," *Int. Conf. Architect. Support Programming Languages Oper. Syst. (ASPLOS)*, Newport Beach, CA, USA, Mar. 5-11, 2011, pp. 291-304.

[17] A. Baumann et al., "The Multikernel: a New OS Architecture for Scalable Multicore Systems," *Symp. Oper. Syst. Principles*, Big Sky, MT. USA, Oct. 11-14, 2009, pp. 29-44.