운영체제의 이식성 향상을 위한 하드웨어 추상화 계층 구조 설계

이동주*, 김지민*, 유민수**
*한양대학교 전자컴퓨터통신공학과
**한양대학교 컴퓨터공학부
e-mail: djlee@rtcc.hanyang.ac.kr, jmkim@rtcc.hanyang.ac.kr, msryu@hanyang.ac.kr

A Structure of Hardware Abstraction Layer for Improving OS Portability

Dong-ju Lee*, Jimin Kim*, Minsoo Ryu**
* Department of Electronics and Computer Engineering, Hanyang University
**Department of Computer Science and Engineering, Hanyang University

요 약

최근 응용 특화된 다양한 구조의 프로세서가 확산됨에 따라 기존 운영체제를 다른 구조의 플랫폼으로 이식하는 비용이 증가하고 있다. 기존 운영체제에서는 소스 코드 수준에서 하드웨어 의존적인 부분을 HAL(hardware abstraction layer)로 구분하여 관리함으로써 이기종 플랫폼간의 이식성을 높이고자 하였다. 그러나 기존 HAL 구조는 대부분 하드웨어의 물리적인 구조만을 고려하여 설계되어 체계적인 이식 작업이 어렵다는 문제점을 가지고 있다. 이를 위해 본 논문에서는 하드웨어의 물리적인 구조와 운영체제의 기능적인 요소를 함께 고려한 HAL 구조를 제안한다. 제안하는 HAL 구조의 효용성은 S3C2410 에서 실행하는 운영체제를 Cell BE 플랫폼으로 이식하는 사례 연구를 통해 검증하였다.

1. 소개

컴퓨팅 성능을 높이기 위한 시도로써 수십~수백개의 코어를 하나의 프로세서에 집적한 many-core 그리고 서로 다른 구조의 코어를 하나의 프로세서에 집적한 이기종 멀티 코어 프로세서 등 다양한 구조의 프로세서가 확산되고 있다. 최근에는 GPGPU 와 같은 응용 특화된 프로세서들이 등장하여 구조의 다양성이 배가되고 있다[4][6].

하드웨어 구조가 다양화됨에 따라 기존 운영체제를 새로운 구조의 하드웨어로 이식하는 비용 역시 증가하고 있다. 기존 운영체제에서는 이식성을 높이기 위해 하드웨어에 직접적으로 의존하는 소프트웨어 계층인 HAL(hardware abstraction layer)[5]을 소스 코드 수준에서 별도로 관리하고 있다.

기존 운영체제의 HAL 은 주로 하드웨어의 물리적인 측면을 반영하여 구조화 하였다. 대표적인 운영체제인 eCos 에서는 HAL 을 Architecture, Variant, Platform 3 가지로 구분하여 구조화하였다. Architecture HAL 은 기본 CPU 구조와 인터럽트 플래그 설정, 문맥 전환, CPU 구동 등을 추상화한다. Variant HAL 은 캐시, MMU 와 FPU 기능들과 같은 CPU 와 연관된 기능들과 온-칩 장치에 의존하는 코드들의 집합을 나타낸다. 마지막으로, Platform HAL 은 플랫폼의 속성들과 플랫폼 구동, 타이머, I/O 레지스터 접근 그리고 인터럽트 컨트롤러를 추상화한다[2]. 가장 널리 사용되는 범용 운영체제 Linux 에서의 HAL 은 각 플랫폼 제조사별로 배포하는 BSP(board support package), 디바이스 드라이버, 어셈블리 코드 등 다양한 형태로 존재한다. Linux 에서는 HAL 계층을 소스 코드 수준에서 개별적으로 관리하고 있지는 않으나, 플랫폼 의존적인 소스 코드의 상당 부분을 별도의 디렉토리로 관리하고 있다[3]. 예를 들어 ARM 프로세서를 내장하고 있는 S3C2410 플랫폼인 경우, CPU 의존적인 부분은 arch/arm 디렉토리에 존재하며 플랫폼 의존적인 부분은 S3C2410 디렉토리에 존재한다.

하드웨어의 물리적인 구조만을 고려한 기존 운영체제의 HAL 은 이종 플랫폼간의 체계적인 이식 작업을 어렵게 하는 문제점을 가지고 있다. 예를 들어 하나의 칩(chip)에 시스템이 내장되는 SoC(System-on-Chip) 구조와 같은 경우 구성 요소들간의 물리적인 경계가 모호해짐으로써 기존 HAL 구조로는 이를 명확히 분류하여 구조화하기 어렵다. 즉, 그래픽 가속기, 캐시, 메모리 등이 CPU 와 함께 하나의 칩에 내장되는 경우 물리적인 경계로 HAL 을 분류하는 전통적인 방법

으로는 HAL 을 효과적으로 관리할 수 없으며 이로 인해 이식 작업의 체계화가 어렵다.

본 논문에서는 서로 다른 하드웨어 플랫폼간의 HAL 구조의 이식성(portability)을 높이기 위해 하드웨어의 물리적인 구조뿐만 아니라 운영체제의 기능적인 측면을 함께 고려한 HAL 구조를 제안한다. 제안하는 HAL 구조의 효용성은 ARM 기반의 s3c2410 프로세서 플랫폼에서 개발된 운영체제를 이기종 멀티프로세서인 PowerPC 기반의 Cell BE(Cell Broadband Engine) 프로세서 플랫폼으로의 이식을 통해 검증하였다.

Cell BE 프로세서는 Sony, Toshiba, IBM 이 공동으로 개발한 프로세서로서 범용 프로세서인 PPE 와 보조 프로세서인 8 개의 SPE 로 구성된 이기종 멀티 코어 프로세서이다. 현재 Cell BE 는 Super computer (IBM Blue gene), HDTV (Toshiba), game console (Sony PS3), acceleration board (Mecury computer system) 등 다양한 분야에 사용되고 있다[4].

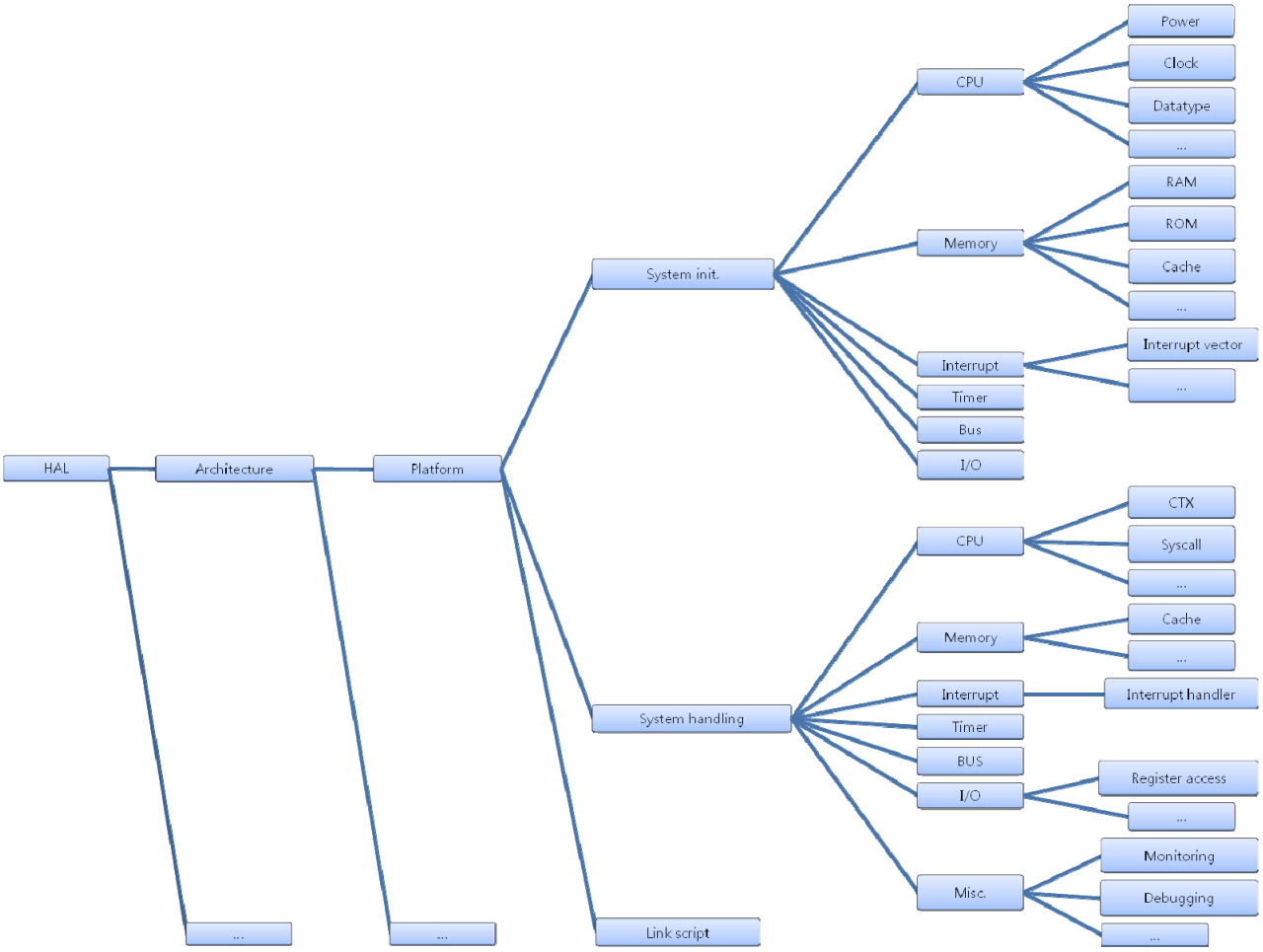
2. 제안하는 하드웨어 추상화 구조

기존 개발된 운영체제를 초기 개발된 하드웨어와 다른 하드웨어 구조로 이식하는 작업은 쉽지 않은 작업이다. 특히, 하드웨어의 구조가 변경될 경우 기존 운영체제를 새로운 구조의 하드웨어에 이식하기 위해서는 HAL 을 전체적으로 재 작성해야 하는 경우가 많기 때문에 많은 비용이 요구된다.

제안하는 HAL 구조는 이식성 향상을 위해 운영체제의 기능적인 요소와 하드웨어의 물리적인 구조를 동시에 고려하였다.

그림 1 은 제안하는 HAL 구조를 나타낸 그림이다. 제안하는 HAL 구조는 프로세서 및 플랫폼에 따라 운영체제의 기능적인 부분을 고려하여 HAL 을 2 가지로 분류하여 구조화한다.

클럭 및 전압 설정, 인터럽트 초기화 등 운영체제 구동에 있어 하드웨어 초기화에 필수적인 요소들은 *System init.*으로 분류하고, 인터럽트 핸들러, 디바이스 드라이버 API 등 운영체제가 구동 중에 하드웨어 접근하기 위해 필요한 필수적인 하드웨어 의존적인 요소를 *System handling* 으로 분류한다. 2 가지로 분류된 각 요소는 하드웨어의 구조에 따라 그림 1 과 같이 다시 분류한다.



(그림 1) 제안하는 HAL 구조

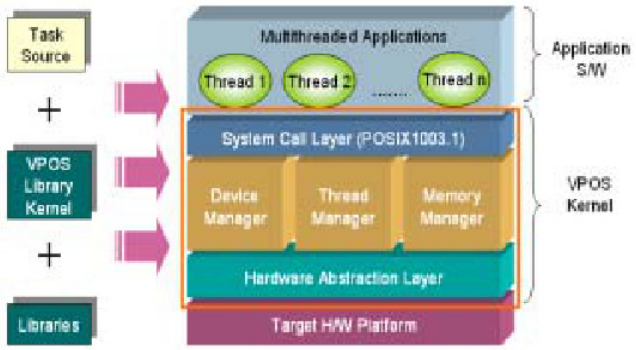
<표 1> 용어 요약

명칭	설명
Architecture	ARM, PowerPC 등 CPU 이름
Platform	제조사가 제조한 플랫폼 이름
System init/cpu	전압, clock 설정, 데이터 타입 설정 등 CPU 초기화에 관련된 코드
System init/memory	캐시, MMU 등의 초기화 코드
System init/interrupt	Interrupt vector 등 interrupt 초기화 코드
System init/timer	타이머의 초기화 코드
System init/bus	플랫폼의 버스 초기화 코드
System init/io	I/O 관련 초기화 코드
System handling/cpu	문맥 전환자 등 운영체제 구동 중에 CPU 에 접근할 수 있도록 하는 handler
System handling/memory	Cache handler, DMA 등 메모리 관련 handler
System handling/interrupt	인터럽트 핸들러
System handling/timer	타이머 핸들러
System handling/bus	Bus 핸들러
System handling/io	I/O 장치 handler 및 레지스터 접근자, 디바이스 드라이버 API 등
System handler/misc	시스템 장치의 모니터링 기능 handler, 운영체제의 디버깅 기능을 위한 핸들러 등
Link script	컴파일 단계에서 하드웨어 구조에 대해 묘사할 수 있는 스크립트

제안하는 HAL 구조는 소스 코드의 디렉토리 구조에 직접적으로 반영된다. 예를 들어 ARM 계열의 s3c2410 프로세서 플랫폼의 clock 초기화 코드의 경우, “hal/arm/s3c2410/system_init/cpu/clock_init.c”와 같이 디렉토리가 구성된다.

3. 사례 연구

제안하는 HAL 구조의 효용성을 입증하기 위해 본 연구실에서 개발한 실시간 운영체제인 VPOS 의 HAL 을 재구조화하여 새로운 하드웨어 구조로 이식하였다.



(그림 2) VPOS 구조

VPOS 는 본 연구실에서 개발한 실시간 운영체제로 계층 구조로 설계되었다. 또한, 정적 우선순위기반 스케줄링, Posix API 호환, Linux 와 유사한 디바이스 드라이버 API 제공 등의 특징을 갖는 초경량 실시간 운영체제이다(그림 2)[1].

ARM 기반의 s3c2410 프로세서에서 동작하도록 작성된 VPOS 는 PowerPC 계열의 HAL 구조를 가지고 있지 않다. Cell BE 프로세서로 이식하기 위해서는 PowerPC 기반의 펌웨어인 open firmware 핸들러를 비롯하여, 부트로더, 인터럽트 핸들러, 메모리 관리자 등 기존 HAL 을 전체적으로 재 작성 해야 한다.

이식작업을 위해 ARM 기반의 s3c2410 프로세서에서 구동되도록 작성된 기존 VPOS 의 HAL 을 제안하는 HAL 구조로 재구조화하였다. 이를 바탕으로 PowerPC 기반의 이기종 멀티프로세인 Cell BE 프로세서로 이식작업을 수행하였다(표 2).

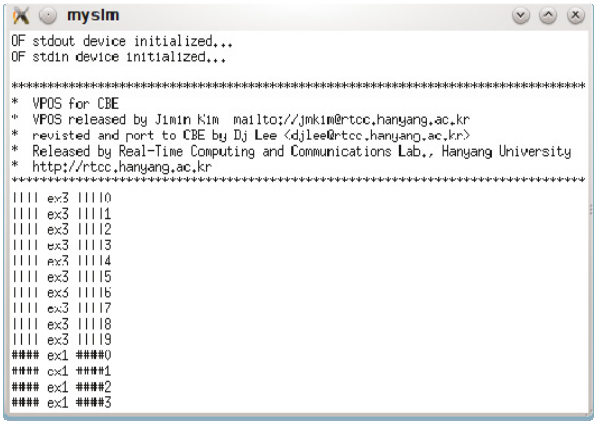
이식 목표인 Cell BE 프로세서 환경은 IBM 에서 제공하는 Cell BE 프로세서 시뮬레이터 Systemsim-cell 을 이용하여 진행하였다[4].

<표 2> s3c2410 구조의 HAL 과 Cell BE 구조의 HAL 예시

분류	S3c2410 파일	S3c2410 파일 설명	Cell BE 파일	Cell BE 파일 설명
System init (Hal/<arch>/<platform>/system_init)	Cpu/hal_arm_s3c2410_cpu.h	cpu 관련 매크로	Cpu/hal_powerpc_cell_cpu.h	cpu 관련 매크로
	Cpu/hal_arm_s3c2410_init_startup.S	시스템의 스타트업 코드	Cpu/hal_powerpc_cell_init_startup.S	시스템의 스타트업 코드
	Cpu/hal_arm_s3c2410_init_cpu_stack.S	스택 맵핑 코드	Cpu/hal_powerpc_cell_init_cpu_stack.S	스택 맵핑 코드
			Memory/hal_powerpc_cell_init_mem_of.h	Open firmware 초기화 인터페이스 및 매크로
			Memory/hal_powerpc_cell_init_mem_of.c	Open firmware 초기화 코드
	Intr/hal_arm_s3c2410_init_interrupt.S	인터럽트 벡터 맵핑 코드	Intr/hal_powerpc_cell_init_interrupt.S	인터럽트 벡터 맵핑 코드
	timer/hal_arm_s3c2410_init_timer.h	타이머 초기화 매크로	Timer/hal_powerpc_cell_init_decrementer.h	Cell BE 프로세서의 타이머는 decrementer 라는 명칭을 사용한다.
	timer/hal_arm_s3c2410_init_timer.c	타이머 초기화 코드	Timer/hal_powerpc_cell_init_decrementer.c	Decrementer (타이머) 초기화 코드
	Io/hal_arm_s3c2410_init_serial.c	시리얼 (GPIO) 초기화 매크로		
	Io/hal_arm_s3c2410_init_serial.c	시리얼 (GPIO) 초기화 코드		
System handling (Hal/<arch>/<platform>	Cpu/hal_arm_s3c2410_handling_cpu.h	Runtime 에 필요한 CPU 관련 매크로	Cpu/hal_powerpc_cell_handling_cpuh	Runtime 에 필요한 CPU 관련 매크로
	Cpu/hal_arm_s3c2410_handling_cpu_ctx.S	문맥 전환자	Cpu/hal_powerpc_cell_handling_cpu_ctx.S	문맥 전환자

/system_handling)			Memory/hal_powerpc_cell_handling_mem_xslation.h	가상 메모리 해석을 위한 매크로 및 인터페이스
			Memory/hal_powerpc_cell_handling_mem_xslation.c	가상 메모리 핸들러
			Memory/hal_powerpc_cell_handling_mem_of.h	Open firmware 인터페이스 및 매크로
			Memory/hal_powerpc_cell_handling_mem_of.c	Open firmware 핸들러
	Intr/hal_arm_s3c2410_handling_interrupt.h	인터럽트 매크로 및 인터페이스 정의	Intr/hal_powerpc_cell_handling_interrupt.h	인터럽트 매크로 및 인터페이스 정의
	Intr/hal_arm_s3c2410_handling_interrupt.c	인터럽트 핸들러	Intr/hal_powerpc_cell_handling_interrupt.c	인터럽트 핸들러
	Timer/hal_arm_s3c2410_handling_timer.h	타이머 인터페이스 정의	Timer/hal_powerpc_cell_handling_decrementer.h	Decrementer (타이머) 인터페이스 정의
	Timer/hal_arm_s3c2410_handling_timer.c	타이머 핸들러	Timer/hal_powerpc_cell_handling_decrementer.c	Decrementer (타이머) 핸들러
	Io/hal_arm_s3c2410-_handling_io_register.h	레지스터 매크로	Io/hal_powerpc_cell_handling_io_register.h	레지스터 매크로
	Io/hal_arm_s3c2410-handling_io_dd.h	디바이스 드라이버 매크로 및 인터페이스 정의	Io/hal_power_cell_handling_io_dd.h	디바이스 드라이버 매크로 및 인터페이스 정의
	Io/hal_arm_s3c2410_handling_io_serial.h	시리얼 (GPIO) 인터페이스		
	Io/hal_arm_s3c2410_handling_io_serial.c	시리얼 (GPIO) 핸들러		

제안하는 HAL 구조로 재 작성된 VPOS 는 이기종 플랫폼으로의 이식 과정에서 표 2 와 같이 직접적이고 명확한 HAL 맵핑이 수행된다는 것을 확인할 수 있었다. 이는 HAL 구조의 변경 없이 Cell BE 프로세서 플랫폼으로 VPOS 를 이식할 수 있다 것을 의미한다. 그림 2 는 Cell BE 프로세서 시뮬레이터인 systemsim-cell 상에서 VPOS 가 3 개의 쓰레드를 구동시키는 화면이다.



(그림 3) Cell BE simulator 에서 구동된 VPOS

4. 결론

기존 운영체제의 HAL 구조는 하드웨어의 물리적인 구조만을 고려하기 때문에 이식성 향상에 어려움을 가지고 있다. 이를 위해 본 논문에서는 하드웨어의 물리적인 구조와 함께 운영체제의 기능적인 면을 고려한 HAL 구조를 제안하였다. 제안하는 HAL 은 현재까지 알려진 대부분의 HAL 코드를 포함하며 명확하고 직관적인 구조를 갖도록 설계되었다. 사례 연구에서는 ARM 기반의 s3c2410 프로세서에 동작하는 실시간 운영체제를 PowerPC 기반의 이기종 멀티프로세서인 Cell BE 프로세서로의 이식을 통해 제안하는 HAL 구조의 효용성을 입증하였다.

참고문헌

[1] 김지민, 유민수, “SoC 설계와 검증을 지원하는 실시간 운영체제,” 한국정보처리학회 춘계학술발표회 논문집, 2005.
[2] eCos, http://ecos.sourceware.org/
[3] Linux, http://www.kernel.org
[4] IBM, http://www.ibm.com
[5] Sungjoo Yoo and Jerraya, A.A., “Introduction to hardware abstraction layers for SoC,” In proceeding of Design, Automation and Test in Europe Conference and Exhibition, 2003.
[6] NVIDIA, http://www.nvidia.com