동적 재구성 가능한 운영체제를 위한 인터럽트 관리 기법

김영필° 송인준 유 혁
고려대학교 컴퓨터학과
{ypkim°, ijsong, hxy}@os.korea.ac.kr

An Interrupt Management for Dynamic Reconfigurable Operating Systems

YoungPill Kim° InJun Song Hyuck Yoo
Dept. of Computer Science and Engineering, Korea University

요 약

하드웨어와 응용 프로그램의 다양한 요구를 만족시키기 위한 운영체제의 재구성 능력이나 기능 확장에 대한 필요성은 최근 들어 급증하고 있다. 운영체제의 재구성을 지원하기 위해서 개선되어야 할 부분 가운데 가장 정적이라고 할 수 있는 부분은 인터럽트 처리와 같은 저수준 입출력 부분이다. 이러한 인터럽트 처리는 하드웨어와 밀접한 부분으로써 운영체제의 설계목적에 따라서 여러 가지 방식으로 구성되어왔으나 각 운영체제에 의존적이며 처리방식이 고정적이라는 한계를 가진다. 본 논문에서는 다양한 형태의 인터럽트 처리 방식을 지원할 수 있는 동적 재구성이 가능한 인터럽트 처리 방식과 그 구조를 제안한다. 제안하는 방식은 커널 컴포넌트들의 동적인 확장과 재구성을 지원하는 커널 컴포넌트 스와핑과 인터포지션 기법을 사용하여 M3K 커널에 구현되었다.

1. 서 론

컴퓨터 시스템의 사용 영역이 기존의 정적인 서버 및 워크스테이션 환경에서 벗어나 PDA, 스마트 폰, MP3 플레이어, 디지털 카메라 등과 같은 다양한 형태의 정보 단말로 확대되고 있다. 이러한 환경에서 응용프로그램과 하드웨어 사이의 인터페이스 역할을 하는 운영체제는 끊임없이 발전하는 하드웨어의 요구와 다양해지는 응용프로그램의 요구를 빠르게 수용할 수 있는 방법을 필요로 하고 있다.

현재 널리 사용되는 대부분의 범용 운영체제는 공평성과 자원 효율의 원칙에 따라서 내부 메커니즘이 결정되며, 특정한 목적으로 사용되는 경우에는 해당 도메인의 필요성에 맞는 운영체제를 사용하게 된다. 일례로 실시간적인 처리를 요구하는 응용 프로그램을 사용하는 경우, 운영체제의 내부 메커니즘들은 실시간적인 처리를 지원할 수 있는 형태로 설계되어야 한다. 또한 널리 사용되고 있는 데이터베이스 전용 시스템의 경우에도 기존의 범용 운영체제가 제공해주는 정책을 사용하면 버퍼 캐쉬 관리 효율이 떨어지게 되므로 도메인에 적합한 메커니즘을 사용해야 한다. 이와 같이 운영체제는 다양한 필요성에 의해 운영체제 서비스가 확장 및 교체 되거나, 서비스의 정책(policy)을 변화시키는 등의 구체적인 방법들이 요구된다.

본 논문에서는 운영체제 기능의 동적인 재구성과 전역적인 확장성에 주안점을 두고 저수준 입출력 처리와 관계된 인터럽트 처리 방식을 동적인 재구성과 확장이 가능하도록 설계하였다. 재구성 및 확장성을 효율적으로 제공하기 위해 M3K[1][2][3][4] 커널에 구현된 컴포넌트 인터포지션(component interposition)과 컴포넌트 스와핑(component swapping)을 사용하였고[5], 인터럽트 처리 요청과 실행을 기존 재구성 기법만을 사용하여 구성하였을 때 발생하는 오버헤드를 최소화하기 위해서 노력하였다.

본 논문의 구성은 다음과 같다. 2장에서는 다양한 목적을 가진 기존 운영체제들에서 사용하는 인터럽트 처리 기법들과 그 공통된 특징들을 기존 연구로써 제시하였다. 3장에서는 본 논문에서 사용한 재구성과 확장성 개념을 인터럽트 처리와 관련하여 정의하고, 4장에서는 동적 재구성이 가능한 인터럽트 처리 메커니즘의 구조와 동작 방식에 대해서 설명한다. 마지막으로 결론에서 본 연구의 결과와 의미를 정리하고, 향후 지속되어야 할 연구 내용에 대해서 논의한다.

2. 관련 연구 및 분석

기존의 운영체제들은 그 목적에 따라서 UNIX나 Linux와 같은 범용 운영체제와 실시간 및 내장형 시스템을 목적으로 하는 운영체제 등 다양한 형태를 가진다. 인터럽트 처리와 같은 저수준 입출력 부분도 그 목적에 따라서 다양한 방식으로 구성되었으며, 본 논문에서는 그러한 방식을 모두 수용할 수 있도록 그 공통적인 특징들을 분류해 보았다.

2.1 인터럽트 우선순위(Interrupt Priority Level)

현존하는 대부분의 범용 운영체제들[6][8][9]나 실시간 내장형 운영체제[7]등은 인터럽트 핸들러나 ISR(Interrupt Service Routine)를 우선순위를 두어 차별화하여 처리하고 있다. 이러한 우선순위는 보통 발생한 인터럽트의 종류에 따라서 할당되며, 보다 낮은 우선순위를 가지는 인터럽트들은 Pending되거나 Nesting된다.

2.2 발생한 인터럽트의 종류(Interrupt Type)

본 논문에서 정의하는 인터럽트는 CPU의 특정 PIN에 전기적인 신호가 오는 것을 의미한다. 이러한 신호는 그 원천에 따라서 하드웨어에 의해 발생한 인터럽트와 트랩(trap) 혹은 예외(exception)과 같이 명령어 처리과정에서 발생할 수 있는 소프트웨어 인터럽트로 구분할 수 있다. 또한 인터럽트 발생 형태에 따라서 커널 혹은 사용자 프로세스가 발생 시점을 예측하기 어려운 비동기(Asynchronous) 인터럽트와 예측 가능한 동기(Synchronous) 인터럽트로 구분해 볼 수 있다. 발생한 인터럽트의 긴급한 정도를 이용하여 분류하면 즉시 처리해야 할 긴급(Critical) 인터럽트와 그렇지 않은 지연가능(Deferrable) 인터럽트로 나누어 볼 수 있다.[9] 그리고 인터럽트 발생의 제어 가능 여부에 따라서 인터럽트 마스킹(Masking)이 가능한 인터럽트(Maskable interrupt)와 그렇지 않은 인터럽트(Non-maskable Interrupt)로 구분할 수 있다.[6][7][8][9]

2.3 인터럽트 실행을 위한 스택(Interrupt Stack)

프로세스 혹은 커널의 실행 컨텍스트(context)와 무관하게 발생하는 인터럽트는 CPU의 레지스터와 플래그 정보들이 저장될 스택이 필요하며, 이러한 스택은 인터럽트 처리만을 위해 커널 안에 별도의 인터럽트 스택을 마련하는 방식과 사용자 프로세스의 커널 모드 스택을 이용하는 방식이 존재한다. 커널 모드 스택 사용 시에는 스택 오버플로의 피해를 막기 위해 스택프레임이 요구된다.[6]

2.4 인터럽트 컨텍스트 전환(Interrupt Context Switching)

인터럽트가 발생 했을 때 스택에 CPU의 상태 정보들이 저장되는 컨텍스트 전환이 일어나야 한다. 보존되는 컨텍스트들은 CPU 레지스터들과 상태 플래그들이 되는데, 모든 정보들을 저장하는 형태(Full context switching)[6][7][8][9]와 동적 코드 생성을 통해서 사용되는 레지스터 정보만을 저장 복구하는 방식(partial context switching)이 존재한다.[10]

2.5 인터럽트의 재진입 가능성(re-entrance)과 중첩(nesting)

인터럽트 서비스 루틴(ISR)들은 재진입 가능하게 처리될 수도 있고[11], 재진입이 가능하지 않은 경우 하나의 인터럽트 서비스 루틴이 처리 된 후에 Pending된 모든 인터럽트들을 처리하는 중첩된 방식[7][9]을 사용할 수도 있다.

3. 동적인 재구성 및 확장성

본 논문에서 사용하는 운영체제의 재구성 및 확장성의 개념은 다음과 같다. 첫 번째로 재구성(Reconfigurability)이란 시스템을 구성하고 있는 요소들이 어떠한 방법에 의해서 새로운 기능을 가지고 있는 다른 모듈로 실행시간에 교체되는 것을 의미한다. 이러한 기법은 시스템의 성능을 향상시킬 수 있으며, 특정한 에러를 포함하는 부분을 바꾸거나, 더 풍부한 기능을 포함하는 서비스로 버전 업(version up) 시킬 수 있다.

두 번째로 확장성(Extensibility)이란 새로운 기능이 기존의 시스템에 추가될 수 있음을 의미한다. 확장성의 개념은 크게 두 가지로 나누어 볼 수 있다. 먼저, 전역적인 확장성은 시스템 관리자나 개발자에 의해서 새로운 서비스들을 추가할 수 있는 특성이며, 이것은 시스템의 모든 사용자들에게 적용된다. 응용프로그램의 확장성은 응용프로그램 개발자가 서비스의 동작방식을 수정하거나 교체하거나 코드를 삽입하는 것이며 이것은 요청한 응용프로그램만이 적용되어야 한다. 다른 응용프로그램들은 기존에 사용하던 서비스들을 그대로 사용 가능해야 한다. 이러한 확장성을 통해서 응용프로그램 자체뿐만 아니라 시스템 전체의 성능 향상을 이끌 수 있다.

3. 재구성 가능한 운영체제를 위한 인터럽트 관리 기법

본 논문에서는 저수준 입출력에 관계된 다양한 인터럽트 처리 방식을 지원하기 위해서 인터럽트 처리에 관련된 루틴을 동적으로 재구성 가능한 커널 컴포넌트로써 구성하였고, 이는 [그림 1]의 구조를 가지는 M3K 커널[1][2][3][4][5]에 구현되었다.

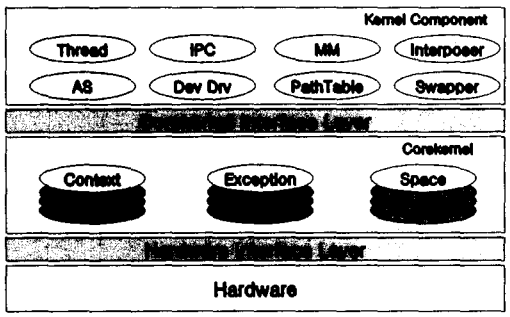

그림 1 M3K 커널 구조

커널 컴포넌트들은 Interposer를 이용하여 커널 서비스들을 동적으로 확장 가능하며, Swapper를 통해 실행 시간에 교체 가능하다. 이는 컴포넌트의 서비스 호출 정보를 트래킹하는 PathTable의 엔트리 정보들을 수정하거나 추가함으로써 구현된다.PathTable을 이용한 컴포넌트 실행은 커널 서비스 호출자가 해당하는 서비스 식별자(service ID)를 이용하여 invokeService() 호출을 통해 이루어진다.[5]

본 논문에서는 이러한 방식에 기반하여 인터럽트 처리와 관련된 정보들을 묶어 [그림 2]와 같이 인터럽트 컴포넌트들을 설계하였다. 인터럽트가 발생하면 발생한 인터럽트의 IRQ번호를 별도로 관리하는 Interrupt PathTable의 서비스 식별자로 사용하여 호출하는 일만을 수행한다. 실선에 해당하는 것은 함수 호출과 같은 Direct Call이고 실선 부분은 Interposer나 Swapper에 의해 관리되어 동적으로 변경 혹은 확장 가능한 서비스 호출 부분이다.

3.1 인터럽트 컴포넌트 (Interrupt Component)

인터럽트 엔트리 컴포넌트에는 인터럽트 우선순위(IPL)와 인터럽트 타입(ITYPE), 스택 주소(STKADDR), 인터럽트 서비스 루틴의 주소(pHandler), 그리고 Corekernel의 Context class로 구현된 인터럽트 컨텍스트들이 포함된다.

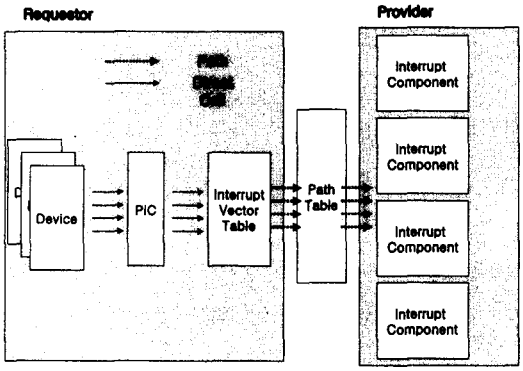

그림 2 PathTable에 기반한 인터럽트 처리 구조

3.2 인터럽트 서비스 루틴 전용 메커니즘

Interposer와 Swapper에 의해 관리되는 PathTable을 이용하여 인터럽트 서비스를 호출하게 되면 메일 박스 IPC(Interprocess Communication)를 이용한 기존의 구현 방식[6]으로는 빠른 응답과 처리를 요구하는 인터럽트 처리에는 적합하지 않다. 그 이유는 커널 컴포넌트들의 실행은 커널 스레드로써 구현되는데 서비스 요청 시점과 실제 처리 사이에는 컴포넌트 스케쥴링을 위한 지연시간이 발생할 수 있기 때문이다. 이를 해결하기 위해서 긴급한 인터럽트 서비스 요청의 경우 Solaris의 Door 메커니즘[11]과 유사하게 스케쥴링 큐에 의존하지 않고 직접 호출하는 방식을 취한다.

4. 결론 및 향후 계획

본 논문에서는 운영체제 기능 가운데 가장 정적이라고 할 수 있는 저수준 하드웨어 입출력에 관계된 인터럽트 서비스 처리 루틴들을 재구성과 확장성을 지원하는 형태의 커널 컴포넌트로 구성하였다. 인터럽트 커널 컴포넌트들은 커널 컴포넌트 인터포지션(interposition)과 커널 컴포넌트 스와핑(swapping) 기법을 이용하여 관리된다. 커널 스레드로 동작하는 인터럽트 처리 컴포넌트간의 전환은 스케쥴러에 의한 지연시간을 극복하기 위해 인터럽트 요청 컴포넌트들의 직접 호출을 사용하였다.

향후 커널 컴포넌트로 구현된 같은 인터럽트 처리 방식에 대해서 실제 구현된 기존 운영체제의 방식과 비교를 하는 것을 연구과제로 삼고 있다.

참고 문헌

[1] 김영호, 고영웅, 아재용, 유혁, "멀티미디어 마이크로 커널 M3K에서 프로세스간 통신 구현 및 성능 분석," 정보과학회 논문지, 2002.

[2] 양순섭, 고영웅, 조유근, 신현식, 최진영, 유혁, "컴포넌트 기반 커널을 위한 프레임워크," 춘계 정보과학회 학술대회 논문집, 1999.

[3] 김영호, 고영웅, 유혁, "M3K에서 IPC 컴포넌트 설계 및 구현," 추계 정보과학회 학술대회 논문집, 2000.

[4] 김영호, 고영웅, 아재용, 유혁, "M3K에서 IPC 구조 및 성능평가," 컴퓨터시스템 연구회 논문집, 2000.

[5] 김영필, 고영웅, 송인준, 김경윤, 유혁, "운영체제의 동적인 재구성을 위한 커널 컴포넌트 관리 기법," 한국정보과학회 컴퓨터 시스템 연구회 추계학술발표회, 2003.

[6] Uresh Valhalia, "UNIX Internals: The New Frontiers," Prentice Hall, 31p~33p, 1996.

[7] Qing Li, "Real-time Concepts for Embedded Systems," CMPBooks, 143p~166p, 2003.

[8] Marshall K. McKusick et al., "The Design and Implementation of the 4.4BSD Operating System," Addison-Wesley, 195p~196p, 1996.

[9] Daniel P. Bovet & Marco Cesati, "Understanding the Linux Kernel 2nd Ed," O'Reilly, 109p~160p, 2003.

[10] Henry Massalin, "Synthesis: An Efficient Implementation of Fundamental Operating System Services," PhD thesis, Columbia University, 1992.

[11] Jim Mauro, Richard McDougall, "Solaris Internals: Core Kernel Architecture," Sun Microsystems Press A Prentice Hall Title, 38p~44p, 459p~461p, 2001.