論文 98-23-3-28

개념 수준의 가상 하드웨어를 이용한 응용 체계 개발 방법론

正會員 고 재 영*, 김 봉 환*, 송 상 섭**

The Methodology for Application System Development Using the Conceptual Level Virtual Hardware

Jae Young Koh*, Bong Hwan Kim*, Sang Seob Song** *Regular Members*

요 약

하드웨어 및 소프트웨어의 개발 과정에서, 서비스 규격의 변화는 하드웨어 개발자에게는 많은 어려움을 준다. 소프트웨어 제작자 또한 하드웨어의 변화에 따른 소프트웨어 규격의 재정립이 필요함으로써 완성될 서비스에 대한 성능의 보장이 어렵다. 이러한 서비스의 변화 요구시 발생되는 문제점을 해결하기 위해서는 개념 수준의 하드웨어를 가상적으로 적용해 볼 필요가 있지만 현재로서는 적용 가능한 특별한 방법이 없다. 본 논문에서는 하드웨어의 개발에서 어플리케이션에 적용하는 일련의 과정에서, 가상 하드웨어를 이용하는 응용 체계 개발 방법론을 제안한다. 제안한 개발 방법론을 하드웨어 및 가상 하드웨어에 적용하여 얻은 실험 결과를 제시한다.

ABSTRACT

In the process of the development of software and hardware, the changes of the service specification distress the hardware developers. The software developers also need to rebuild the software according to the changes of the hardware specification. Therefore, the frequent changes of the specification may not guarantee the quality of service required. To overcome these conditions, applying the conceptual level hardware to the existing application and the process of new application system development is essential. But there is no special way to do so. In this paper, we introduce the methodology for application development using the conceptual level hardware which can be applied a series of jobs to the process of the application development. In addition, we make an experiment in the real application with the real and virtual hardware by using the devised development methodology.

Ⅰ. 서 론

최근에는 Windows 95 및 Windows NT가 급속도로 보급되면서 일반화되고 전산망의 주된 운용 환경을 제공하고 있다. 특히 본 연구의 기반 운영 체제인 Windows NT는 기타 다른 운영체제가 가지는 구조와 달리 마이크로 커널(Micro Kernel) 개념을 바탕으로 운영

*국방과학연구소
**전북대학교 전기·전자·제어공학부
論文番號:97426-1126
接受日字:1997年 11月 26日

체제의 모든 기능을 드라이버들의 집합으로 구성하여 서로 각기 다른 권한을 가지도록 설계되어 있다[1]. 특히 하드웨어를 통한 입력 및 출력은 특정한 권한을 가지는 드라이버를 통해서만 가능하도록 지원하고 있다. 즉, 새로운 하드웨어를 제작하여 전산기에 부착하기 위해서는 제작한 하드웨어에 대한 디바이스 드라이버를 만들어 Windows NT에 등록한 후 그 드라이버를 통해서만 입출력의 수행이 가능하다[2][3].

현재의 개발 방법은 API 규격을 제정하고 하드웨어별, 소프트웨어 별로 일부 단계까지는 병행하여 개발을 진행할 수 있으나, 전체적인 개발 과정으로 보면 순차적인 개발이라 할 수 있다. 현재의 개발 방법의 가장 큰 문제점은 개발 초기단계에서 완성될 응용 체계에 대한 확실한 성능의 보장을 예측할 수 없다는 것이다.

본 논문에서는 응용 체계의 구현에 소요되는 시간을 단축시키고, 개발 초기단계에서 완성될 응용 체계에 대한 성능을 보장하며, 요구 규격의 변화에 능동적으로 대처할 수 있는 가상 하드웨어를 이용한 개발 방법을 제안한다. 하드웨어를 대신하는 가상 하드웨어는 소프트웨어 모듈이며, 파라메터를 설정하여 하드웨어의 기능을 수행한다. 하드웨어의 규격 변경은 파라메터의 변경으로 가능하여 실시간으로 응용 소프트웨어와 연동하여 성능을 평가할 수 있다. 제안한 개발 방법론을 적용하면 응용 체계의 운용 환경에 최적인 하드웨어 규격을 제작하기 이전에 알 수 있어 경제적인 응용 체계를 구축할 수 있다.

가상 하드웨어에 대한 디바이스 드라이버 형태의 소프트웨어를 가상 디바이스 드라이버(Virtual Device Driver, VDD)라 한다. VDD를 제작하기 위해서는 첫 번째로 전산기에 적용되는 하드웨어 및 응용 체계의 특성을 분석하여 개념 수준의 일반적인 형태의 가상 하드웨어를 만들고, 두 번째로 가상 하드웨어가 실질적인 하드웨어로서의 기능(Function)을 수행할 수 있도록 파라메터를 설정하며, 끝으로 이를 적용하려는 전산기의 운영체계와 드라이버 환경에 맞추어 프로그램 한다. 제작한 VDD는 API를 이용하여 응용 소프트웨어를 만드는 프로그래머 입장에서는 실제 하드웨어와 동일하게 인식되어 여러 종류의 응용 체계에 투명하게 적용된다.

본 논문의 구성은 기존의 응용 체계 개발 과정을 기술하고, 가상 하드웨어를 이용한 개발 방법을 위하여 Windows NT의 일반 사항과 NT 드라이버의 특성을 설명하고, 하드웨어 디스크립션에 대하여 기술한다. 기존의 Windows NT 디바이스 드라이버의 구성 형태와 여러 개의 표본 드라이버를 분석하여 VDD를 제작하고, 파라메터를 설정하여 실제 하드웨어와 같은 동작 특성을 보이고, 실험 결과를 바탕으로 임의의 하드웨어에 대해서도 가상 하드웨어를 이용한 개발 방법을 적용할 수 있음을 보인다.

Ⅱ. 기존의 응용 체계 개발

기존에 운용되고 있는 응용 체계에 하드웨어를 추가하고 연동하여 새로운 서비스를 구축하기 위해서는 하드웨어와 API가 필요하다[4][10][11]. 이러한 일련의 작업을 통하여 원하는 서비스를 실현하는 것은 결코 쉬운 일은 아니다. 또한 개발 과정을 살펴보면, 하드웨어 제작, 디바이스 드라이버 제작, API 제작 및 응용 소프트웨어 제작이 각기 순차적으로 이루어지고 있다. 이러한 이유로 개발 기간을 단축은 물론, 실제 응용 체계에 무리 없이 적용할 수 있다는 판단을 개발 전이나 개발 단계에서 내리는 것은 한계가 있다. 순차적인 개발 과정이 단순한 형태의 전산 장비에서는 무리 없이 적용될 수 있으나, 컴퓨터 시스템이 날로 복잡해지고 다양해지는 상황에서는 하드웨어, 디바이스 드라이버, API 및 응용 소프트웨어간의 관계가 상호 복잡하게 연관되어 적용하기 어렵다. 또한 컴퓨터 시스템의 발전과 더불어 다양해지고 있는 요구를 응용 체계에 능동적으로 적용하는 것은 어

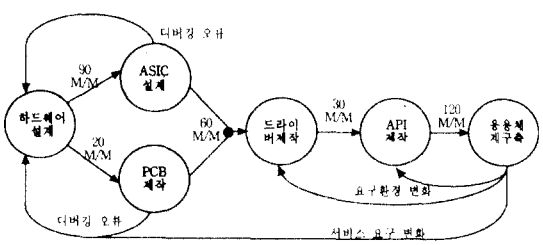


그림 1. 기존의 응용 체계 개발 과정(순차진행)
Fig. 1 The existing development procedure of application system(sequential progress)

려운 실정이다. 현재 하드웨어가 제작 단계에 있거나 하드웨어를 제공하지 못하는 상황에서 응용 소프트웨어의 제작은 불가능하며, 완성될 응용 체계에 대한 성능을 보장할 수 없다. 또한 완료된 응용 체계의 성능이 미비하여 하드웨어 규격 및 소프트웨어 규격의 변경이 필요하게 되면, 개발 초기단계부터 개발을 다시 진행해야 되어 추후 완성단계까지 소요되는 기간은 예측할 수 없다. 그림 1과 그림 2는 기존의 응용 체계 개발 과정 및 구성 도이다.

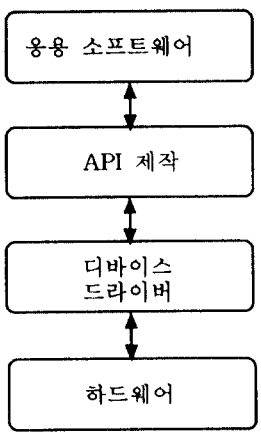


그림 2. 기존의 응용 체계 구성도
Fig. 2 The existing application system configuration

Ⅲ. 가상 하드웨어를 이용한 개발

응용 체계의 구현에 소요되는 시간을 단축시키고, 개발 단계에서 완성될 응용 체계에 대한 성능을 보장하며, 요구 규격의 변화에 능동적으로 대처할 수 있는 개발 방법이 필요하다. 이를 위하여 하드웨어를 대신하는 소프트웨어 모듈을 구성하고 파라메터를 설정하여 하드웨어의 기능을 수행하도록 하는 개발 방법을 제안한다.

파라메터의 설정은 크게 실행 속도 및 지연 요소로 구성되어 있으며, 지연 시간 요소를 분석하여 그에 따른 처리속도에 대하여 가상 하드웨어를 제작한다. 가상 하드웨어는 디바이스 드라이버 형태로 제작하여 응용 체계에 적용한다. 가상 하드웨어에 대한 디바이스 드라이버 형태의 소프트웨어를 VDD라 한다. VDD를 제작하기 위해서는 첫 번째로 전산기에 적용되는 하드웨어 및 응용 체계의 특성을 분석하여 개념 수준의 일반적인 형태의 가상 하드웨어를 만들고, 두 번째로 가상 하드웨어가 실질적인 하드웨어로서의 기능을 수행할 수 있도록 파라메터를 설정하며, 끝으로 이를 적용하려는 전산기의 운영체계와 드라이버 환경에 맞추어 프로그램 한다. 제작한 VDD는 API를 이용하여 응용 소프트웨어를 만드는 프로그래머 입장에서는 실제 하드웨어와 동일하게 인식되어 여러 종류의 응용 체계에 투명하게 적용된다. 하드웨어의 규격 변경은 파라메터의 변경으로 가능하여 실시간으로 응용 소프트웨어와 연동하여 성능을 평가할 수 있다.

실존하지 않는 하드웨어에 대하여 정량적으로 기술할 수 있다면, 가상의 하드웨어에 대한 하드웨어 디스크립션(Hardware Description, HD)을 구성할 수 있으며, 이를 바탕으로 하드웨어를 대신할 수 있는 가상 하드웨어를 만들 수 있다(그림 3). 가상의 하드웨어를 만들기 위해서는 각종 HD를 분석하고 분류하고 정형화하는 과정을 거쳐야 한다.

운용되고 있는 응용 체계에 새로운 서비스의 연동을 위해 하드웨어를 설계하는 사람은 하드웨어에 대한 HD를 구성하여 가상 하드웨어를 만들 수 있으며, 응용 체계에 적용하여 어느 정도의 성능을 유지할 수 있는 지를 실험할 수 있다. 가상 하드웨어를 이용한 개발 방법을 통해 응용 체계의 요구 변화에 따라 가상 하드웨어의 파라메터 변경을 반복하면서 안정적이고 최적의 파라메터를 추출하면, 실제 제작할 하드

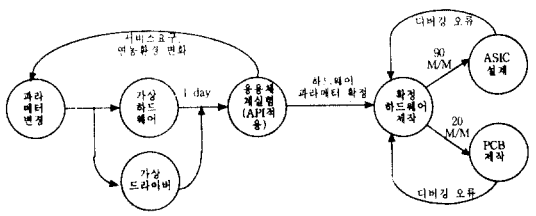


그림 3. 가상 하드웨어를 이용하는 개발 모형을 적용한 개발과정
Fig. 3 The development procedure of application system applying a virtual hardware

웨어의 부품 선정 및 인터페이스 규격 등에 도움을 준다.

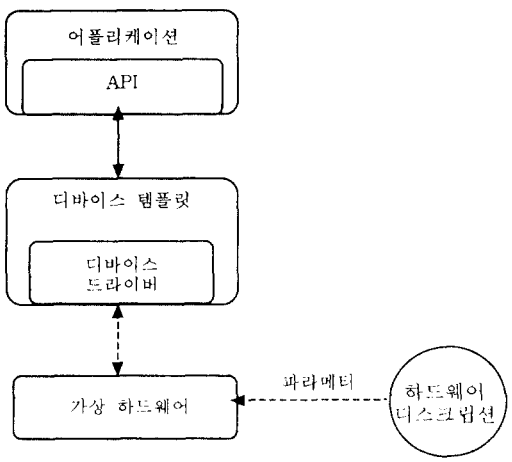


그림 4. 가상 하드웨어를 이용한 개발 방법론
Fig. 4 The development methodology using a virtual hardware

3.1 Windows NT 및 Windows NT 드라이버

Windows NT는 명료한 드라이버 모델(Driver Model)을 제공한다. NT 드라이버는 원시 코드(Source Code)의 수정 없이 다양한 하드웨어에서 수행할 수 있도록 호스트 머신에 대한 상세한 사항을 추상화시켰다. 이러한 추상화의 많은 부분은 NT HAL(Hardware Abstraction Layer, 하드웨어 추상 계층)안에 들어 있으며 다양한 하드웨어 플랫폼(Platform)을 제공하기 위해서 다양한 HAL이 존재한다. 일 예로, 드라이버 내부는 HalGetInterruptVector라는 커널 API를 이용하여 인터럽트 객체(Object)를 얻은 후에 IoConnectInterrupt라는 커널 API를 이용하여 인터럽트 객체를 전달함으로써 ISR(Interrupt Service Routine, 인터럽트 서비스 루틴)을 설정할 수 있다[3][6].

NT 구조의 중요한 부분은 계층화된 드라이버 모델의 사용이다. 이러한 모델은 사용자 어플리케이션과 실제로 제어되는 하드웨어 사이에서 드라이버 스택(Driver Stack)을 제공하여 여러 계층으로 구성된다. 추가할 새로운 기능들은 기존의 드라이버 스택에 새로운 드라이버를 삽입함으로써 추가된다. NT 코드는 사용자 모드(User Mode)와 커널 모드(Kernel Mode) 중 하나의 모드에서 수행된다. Win32 어플리케이션은 전적으로 사용자 모드에서 수행되는 반면, 대부분의 NT 드라이버는 커널 모드에서 수행된다. Intel X86 플랫폼에서 사용자 모드는 링(Ring) 3 코드로서 구현되며, 커널 모드는 링 0 코드로 구현된다.

Windows NT의 커널은 유닉스 등 다른 운영체제가 가지는 커널 구조와 달리 분산 운영체제(Distributed Operating System)가 가지는 마이크로 커널(Micro Kernel) 구조를 가진다. 운영체제는 디바이스 드라이버들의 집합이며, Windows NT의 모든 기능은 운영체제를 구성하는 드라이버 기능의 집합이다. 하드웨어의 입출력을 통한 새로운 서비스 구축은 하드웨어를 제작하여 컴퓨터에 설치하고, 디바이스 드라이버를 제작하여 Windows NT에 등록한 후 디바이스 드라이버를 통하여 가능하다.

NT 드라이버는 계층적 드라이버(Layered Driver)와 일체형 드라이버(Monolithic Driver)로 구분될 수 있다. 일체형 드라이버는 드라이버 상단에서 하드웨어를 제어하고, 하단에서 사용자 모드 인터페이스를 제공하는 드라이버이다. 본 논문에서 제시되는 모든 형태의 드라이버는 일체형 드라이버를 중심으로 구성된다.

3.2 API

API는 디바이스 드라이버를 통하여 하드웨어를 이용할 수 있도록 응용 체계에 동적 링크 라이브러리(Dynamic Link Library, DLL)의 형태로 제공된다. 윈도우즈 운영체제의 특징 중의 하나는 DLL을 사용할 수 있다는 점을 들 수 있다. 이름에서 알 수 있듯이 DLL은 응용 체계의 실행 시점에서 동적(Dynamic)으로 링크(Link)되는 라이브러리이다. 일반적으로 .LIB라는 확장자를 가지는 정적 링크 라이브러리(Static Link Library)는 컴파일 단계에서 링크 되는 반면, DLL은 실행 시점에서 응용 체계에 링크 됨으로써 메모리의 효율적인 관리에 이점을 준다[4][8][10][11].

3.3 하드웨어 디스크립션

HD는 하드웨어의 구성 특성을 분류하여 정형화한 형태로 하드웨어가 가지는 여러 가지 특성을 추출한 모음이다. 가상 하드웨어는 HD의 특성에 적절히 응

답할 수 있는 기능을 가지고 있어야 하는 조건을 가진다. 여기서 "적절히"의 의미는 가능한 한 실제 하드웨어에 가깝도록 응답한다는 뜻으로 실제 하드웨어의 고유한 속도나 데이터 통신 량 등은 논외로 한다. 이 사항은 하드웨어 개발자에 의해 충분히 고려될 수 있는 부분이며, 소프트웨어로 구현해도 절대치의 속도는 기대할 수 없음을 밝힌다. 본 논문에서는 하드웨어에 대한 HD로서 하드웨어의 실행 속도 및 지연 시간 등에 대한 행동 특성을 정형화한 파라메터로 구성하였다. 지연 시간이란 어플리케이션에서 하드웨어를 통하여 원하는 서비스를 요구할 때, 응용 체계에서 시작하여 하드웨어에서 그 기능을 수행하고 서비스 결과를 돌려주는 시점까지의 시간을 말한다. 그림 5는 응용 체계에 특정 서비스를 연동하는 구조를 나타낸다.

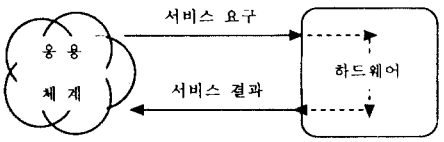


그림 5. 하드웨어의 서비스 연동 구조
Fig. 5 The interoperatin structure of service of hardware

하드웨어의 지연 시간을 알기 위해서는 일정량의 데이터를 처리하는 데 걸리는 시간 즉, 처리 속도에 대한 파라메터를 구체화할 필요가 있다. 하드웨어의 기능 수행시 처리 속도에 영향을 주는 요소들은 일반적으로 그림 6과 같다.

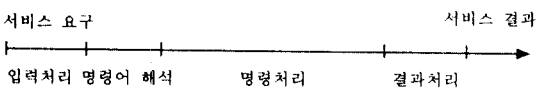


그림 6. 일반적인 하드웨어의 처리지연 요소
Fig. 6 The general hardware's processing delay elements

Ⅳ. 실험 및 분석

제안한 가상 하드웨어를 이용한 응용 체계 개발 방법론을 실험을 통해 입증하기 위하여 하드웨어를 제작하여 응용 체계에 적용하였다.

4.1 가상 하드웨어 구현

가상 하드웨어는 크게 5가지 루틴으로 구성될 수 있으며 다음과 같다.

4.1.1 드라이버 엔트리 루틴

드라이버가 NT에 의해 적재될 때, 드라이버의 DriverEntry 루틴은 NT에 의해 호출된다. 입출력 요구 패킷(I/O Request Packet, Irp)은 NT 디바이스와 통신하거나 NT 디바이스 상호간의 통신을 위한 기본적인 방법이다. Irp는 크게 2 부분으로 구성되는데 Irp 스택과 Irp 자체가 그것이다. Irp 스택은 Irp가 계층과 계층사이를 통과할 때 드라이버와 드라이버 사이에서 변화될 수 있는 정보를 가지고 있다. Irp 자체는 Irp의 생존 기간을 통하여 변할 수 없는 정보를 가지고 있다. Irp 스택은 pIrpStack→MajorFunction에 위

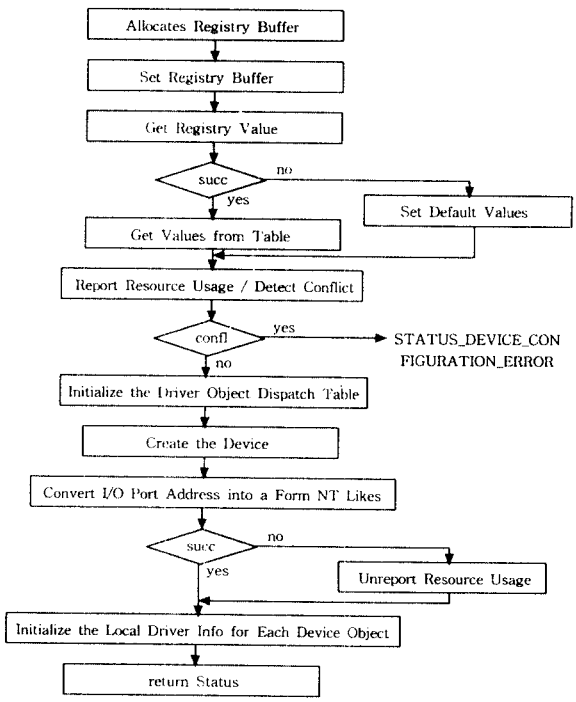


그림 7. 드라이버 엔트리 루틴의 동작 흐름도
Fig. 7 The flow diagram of DriverEntry routine

치한 주 기능 형식(Major Function Type)을 가진다.

이 루틴에서는 디바이스 객체를 생성하고 드라이버 객체에 분기 엔트리 포인터를 설정하여 NT의 모든 요청이 지정된 루틴에게 전달되도록 한다. 드라이버 엔트리 루틴의 함수 형식은 다음과 같으며 동작 흐름은 그림 7과 같다.

```
NTSTATUS DriverEntry
    (IN PDRIVER_OBJECT DriverObject,
    IN PUNICODE_STRING RegistryPath)
```

4.1.2 디바이스 객체 생성 루틴

객체는 자원이나 엔터티(Entity)를 표현하는 불분명하거나 부분적인 데이터 구조와 그 데이터 구조를 대상으로 원하는 명령을 수행하는 명령어 집합(Instruction Set)을 가지고 있다. 심볼릭 링크는 Win32 어플리케이션이 드라이버에 대한 접근을 얻기 위하여 CreateFile 함수 호출에서 사용하는 이름이다. 각각의 디바이스 객체는 하나의 디바이스 확장(Device Extension)을 갖는다. 디바이스 호출에 관련되어 지속적으로 유지되어야 하는 모든 변수들은 디바이스 확장에 저장된다. 일반적으로 모든 디바이스 정보는 드라이버가 접근할 수 있도록 디바이스 확장에 보존된다.

본 루틴에서는 디바이스 객체를 만들고 \DosDevices 안에 있는 심볼릭 링크를 만든다. 심볼릭 링크는 디바이스 이름과 \DosDevices 안의 이름으로 만들어진다. 이로써 Win32 어플리케이션이 그 디바이스를 사용할 수 있다. 디바이스 객체 생성 루틴의 함수 형식은 다음과 같으며 동작 흐름은 그림 8과 같다.

```
NTSTATUS PCreateDevice
    (IN PWSTR PrototypeName,
    IN DEVICE_TYPE DeviceType,
    IN PDRIVER_OBJECT DriverObject,
    OUT PDEVICE_OBJECT *ppDevObj)
```

4.1.3 디스패치 루틴

드라이버에 대한 디스패치 핸들러로서 모든 IRP를 처리할 책임을 진다. 아규먼트는 디바이스 객체에 대한 포인터와 현재 IRP에 대한 포인터이다. 특히, 다중 프로세싱 환경에서 수행하는 드라이버를 구축하기 위해, 드라이버 안에서 어플리케이션을 구별할 수 있는 자료 구조를 사용하였다. 디스패치 루틴의 함수 형식은 다음과 같고, 동작 흐름은 그림 9와 같다.

```
NTSTATUS PDispatch(IN PDEVICE_OBJECT pDO,
    IN PIRP pIrp)
```

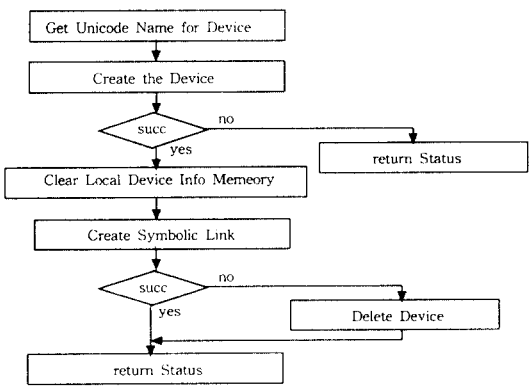


그림 8. 디바이스 객체 생성 루틴의 동작 흐름도
Fig. 8 The flow diagram of CreateDevice routine

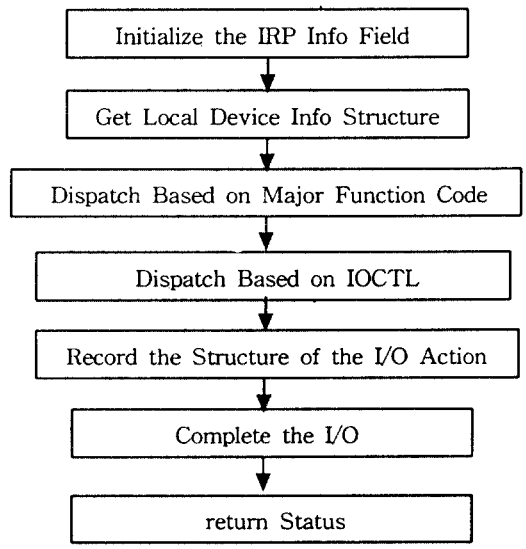


그림 9. 디스패치 루틴의 동작 흐름도
Fig. 9 The flow diagram of Dispatch routine

4.1.4 언로드 루틴

현재 사용중인 드라이버를 언로드 한다. DriverEntry 또는 드라이버의 수행 중에 할당되었던 모든 자원을 해제한다. 언로드 루틴의 함수 형식은 다음과 같으며 동작 흐름은 그림 10과 같다.

VOID PUnload(PDRIVER_OBJECT DriverObject)

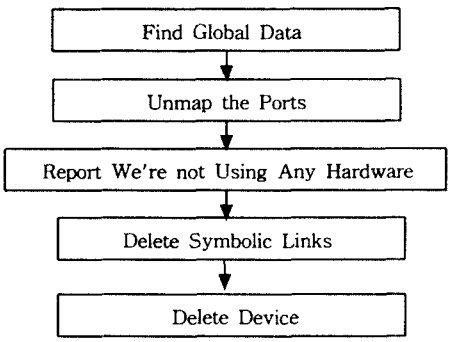


그림 10. 언로드 루틴의 동작 흐름도
Fig. 10 The flow diagram of Unload routine

4.1.5 내부 지원 함수들

드라이버의 세부 기능을 정의하는 내부 함수이다. 내부 지원 함수의 형식은 다음과 같으며 동작 흐름은 그림 11과 같다.

NTSTATUS IoctlWritePort
(IN PLOCAL_DEVICE_INFO pLDI,
IN PIRP pIrp)

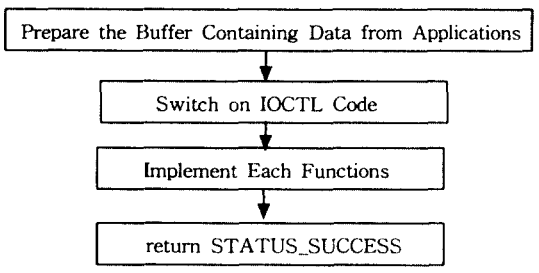


그림 11. 내부 지원 함수의 동작 흐름도
Fig. 11 The flow diagram of internal supporting functions

4.2 하드웨어 디스크립션 분석

4.2.1 입력 처리 지연 시간

입력 처리 지연 시간은 일반적으로 전체 지연 요소에 비하여 상당히 작다. 버스 전달 지연 시간, 버퍼링 및 프로토콜 수행 지연 시간으로 분류한다. 전산기의 버스 전달 시간은 100Mbps 이상의 속도를 낼 수 있으므로, 하드웨어의 지연 시간에 대한 영향은 적은 편이다.

버스 지연 시간(Tbus) = 1/버스타입(Cbustype)

버퍼링 및 프로토콜 수행 지연 시간은 버퍼의 길이에 따라 상당한 영향을 미치며, 수십 바이트에서 수십 킬로(K) 바이트까지를 선택 범위로 한다. 프로토콜은 버퍼를 관리하고 입출력에 대한 판단 근거를 마련하는 등의 일련의 작업에 소요되는 요소로써 상당히 미약하며 파라메터 Cproto는 CPU의 속도에 비례한다.

프로토콜 처리시간(Tproto) = 데이터 길이(Cdatalen)
* 데이터 1bit 처리속도(1/Cproto)

4.2.2 명령어 해석 지연 시간

명령어 해석 지연 시간은 각 명령어에 따라 다르게 표현될 수 있으며, 그에 따라 속도가 달라 진다. 명령어 해석은 헤더를 분석하는 시간에 비례하여 헤더 처리 시간과 구현 방법에 따라 영향을 받는다. 구현 방법은 C 언어와 어셈블러 구현 두 가지 방법이 있다고 가정한다. 파라메터의 구성은 CPU 의 속도에 비례하는 상수로 구성될 수 있다.

명령어 해석 지연시간(Tcmd)
= 헤더 처리 시간 * 구현 방법
= 헤더 길이 * (1/헤더 1바이트 처리속도) * 구현 방법
= Chdrlen * (1/Chdrproc) * Cimp

4.2.3 명령 처리 지연 시간

명령 처리 지연 시간은 일정량의 데이터를 처리하는데 걸리는 시간을 말하며 대부분의 지연 시간이 발생하는 부분으로 다음과 같은 요소들로 구성될 수 있다.

4.2.3.1 알고리즘 종류와 구현 방법 및 CPU 최대 사

이클 수에 따른 지연 시간

알고리즘 종류는 각 알고리즘이 원하는 수준의 서비스를 유지하면서 특정 알고리즘을 처리에 소요되는 지연 시간이다. 실제 알고리즘 구현시의 파라메터는 특정 알고리즘을 실행시킴에 있어 1비트 데이터를 처리하는데 소요되는 평균 사이클 수(Cycles Per Instruction, CPI)로 나타낸다. 알고리즘 구현 방법에 따른 지연 시간은 크게 C 또는 ASM으로 구현한 바에 따라 구분할 수 있다. CPU 처리 지연 시간은 CPU 속도가 하드웨어의 전반적인 성능에 크게 영향을 미치는 요소로서 CPU의 최대 싸이클 수로 나타낸다. 알고리즘 종류와 구현 방법 및 CPU 최대 사이클 수에 따른 지연 시간은 필요 알고리즘 사이클 수를 CPU 최대 사이클 수로 나눈 다음 알고리즘 구현 방법을 곱한 것과 같다.

알고리즘 지연시간(Talgo)
=(Calgcycle/Ccpucycle)*Calgimp

예를 들면, 1 bit를 150 사이클이 필요한 알고리즘을 30Mcycle 속도를 가지는 CPU를 통하여 수행할 때는 1/200,000 초에 1 bit를 처리 할 수 있다.

4.2.3.2 하드웨어의 입출력 지연 시간

하드웨어의 입출력 속도(Cmodiospd) 차이는 하드웨어 속도에 상당한 영향을 준다.

하드웨어 입출력 지연 시간(Tmodio)=1/Cmodiospd

하드웨어의 입출력 여부는 입출력 수행을 판단하는 요소로서, 입출력을 수행하면 1이고 그렇지 않으면 0이다. 수행 여부 변수(Boolean) 인 Sifmodio는 1 또는 0을 가진다. 하드웨어의 입출력 횟수는 전체 처리 속도에 영향을 준다. 하드웨어의 입출력 횟수Cmodionum)는 0 이상의 자연수 있다. 또한 하드웨어의 입출력을 위한 프로토콜 오버로드(Cmodioproto)는 하드웨어 입출력 지연 시간을 초과하지 않는다고 가정한다.

하드웨어 입출력 지연 시간=
((디바이스 내 입출력 속도 + 프로토콜 오버로드)
*입출력 횟수)*입출력 여부

Tmodio=[(Cmodiospd +Cmodioproto)*Smodionum]
*Sifmodio
=[{(1/Cmodiospd) +Cmodioproto}*Smodionum]
*Sifmodio

4.2.3.3 알고리즘 수에 따른 알고리즘 교체 지연 시간

하드웨어에 내장되는 알고리즘 수는 하드웨어가 필요로 하는 메모리 양을 의미한다. 이는 알고리즘 교체 지연 시간의 존재 여부를 결정하는 요소로서 2개 이상이면 알고리즘 교체 지연 시간이 서비스 형태에 따라 추가될 수 있다. 만약 내장 알고리즘 수가 2 이상이면 처리시간 변경 여부(Cifswitch)는 1(True)로 설정된다.

알고리즘 교체 지연 시간은 하드웨어 내에 알고리즘이 여러 개가 대상되어 있는 상황에서, 한 알고리즘이 메모리에 적재되어 실행되고 있는 상태에서 다른 알고리즘의 요구에 의해 대상 알고리즘이 메모리에 적재되고 스위칭되는 시간을 의미한다. 이 파라메터는 내상 알고리즘 수가 1일 때는 적용되지 않는다.

알고리즘 교체 지연 시간
=대상 알고리즘의 메모리 크기*CPU 처리 시간

표 1. 파라메터
Table 1. Parameters

파라메터	내 용	실험 설정치
Cbustype	버스 타입	30Mbps(ISA)
Chdrlen	명령어 헤더 길이	20byte
Chdrproc	명령어 헤더 처리 시간	30Mbps
Calgcycle	비트 처리 당 필요 사이클 수	150cycle
Calgimp	알고리즘 구현 방법	1
Ccpucycle	CPU의 최대 사이클 수	30Mcycle
Cmodiospd	하드웨어의 입출력 속도	9.6Kbps
Cmodioproto	하드웨어의 프로토콜 오버헤드	1sec
Cmodionum	하드웨어의 입출력 횟수	1
Cifmodio	하드웨어의 입출력 여부	1
Cmemlen	처리루틴 프로그램 크기	2Kbyte
Cmemtime	메모리 적재 시간	10Mbps
Cifswitch	처리루틴 변경 여부	1

*(읽는 시간 +전송 시간 +쓰는 시간)

Tld = Cmemlen * Tcpu * (Tmrd +Ttrs +Tmwr)

= Cmemlen * (1/Ccpucycle) * {(1/Cmrd) +(1/Ctrs) +(1/Cmwr)}

Cmemlen : 알고리즘이 차지하는 메모리 크기(byte)

Tcpu : CPU 처리 지연 시간

Tmrd : 메모리에서 1 바이트 읽는 시간

Ttrs : 메모리에서 목적 메모리로 1 바이트를 전송하는 시간

Tmwr : 목적 메모리에 1 바이트를 쓰는 시간

Ccpucycle : CPU 처리 속도

Cmrd : 메모리에서 1 바이트 읽는 속도

Ctrs : 메모리에서 목적 메모리로 1 바이트를 전송하는 속도

Cmwr : 목적 메모리에 1 바이트를 쓰는 속도

4.3 실험 환경

본 논문에서는 VDD의 전체적인 성능을 분석하기 위해서, 실제 하드웨어의 각종 기능을 처리하는데 소요되는 지연 시간을 추출하여 파라메터로 설정하였다. 그림 12는 실험 방법에 대한 개념을 나타내고 있다.

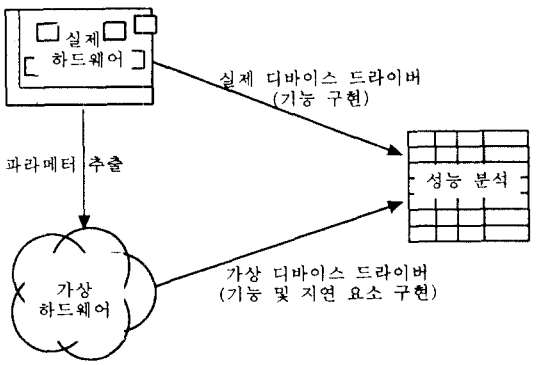


그림 12. 실제 하드웨어와 가상 하드웨어의 실험 방법

Fig. 12 The experiment method of real and virtual hardware

실험을 위해 실제 및 가상 드라이버의 개발 도구는 WinDK DDK Class Library와 Windows NT DDK (for Windows NT Workstation 4.0), Win32 SDK(for Windows 95 and Windows NT Workstation 4.0)를 사용하고, 통합 개발 환경은 Microsoft Visual C++ 5.0을 사용한다[3][7][9].

실제 하드웨어는 스마트카드를 이용하여 PC의 접근 통제를 수행하는 장치이며, 구성은 그림 13과 같다. PC와의 통신은 ISA 버스를 통하여 이루어진다.

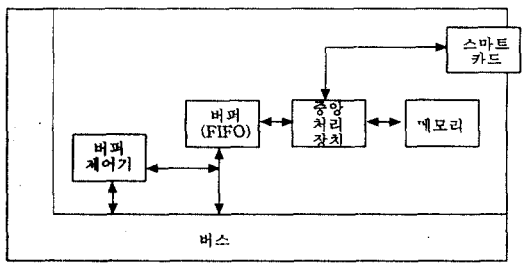


그림 13. 실제 하드웨어의 구성

Fig. 13 The real hardware configuration

디바이스 드라이버 및 VDD를 이용한 실험 환경은 다음 그림 14와 같으며 구현된 디바이스 드라이버는 Windows NT 4.0 Workstation의 레지스트리에 등록되어 시스템이 부팅될 때 운영체제가 인식할 수 있도록 하였다. 기능 구현은 DLL 및 각각의 디바이스 드라이버에서 구현되었다.

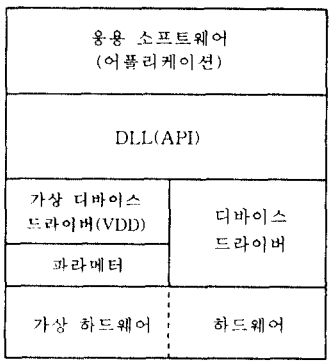


그림 14. 실제 및 가상 하드웨어의 실험 환경

Fig. 14 The experimental environment of real and virtual hardware

그림 14의 DLL은 어플리케이션에서 호출할 수 있는 함수를 구현한 것으로 Windows NT와 같은 다중 프로세싱 환경에서 여러 어플리케이션이 공유할 수 있는 기반을 제공한다. 본 논문에서는 실제 하드웨어를 이용하는 어플리케이션과 가상 하드웨어를 이용

하는 어플리케이션이 함께 사용하고 있다. 또한 각각의 어플리케이션을 다중 인스턴스에 대해서도 동일하게 공유될 수 있다[8]. 디바이스 드라이버는 다중 프로세싱 환경에 부합하도록 내부적으로 간단한 데이터 구조를 이용하여 구현하였다. 즉 하드웨어에서 처리할 수 있는 프로세서 개수에 대한 인덱스 어레이를 구성하여 할당된 프로세서 식별 자와 그에 해당하는 어플리케이션에 대한 관리를 할 수 있도록 하였다. 또한 상호 배제(Mutual Exclusion)를 달성할 수 있도록 자료구조를 구성하였다[1].

4.4 실험 결과 및 분석

본 논문에서는 파라메터를 크게 실행 속도 및 지연 요소로 구성하였으며, 지연 시간 요소를 분석하여 그에 따른 처리 속도에 대하여 실험을 하였다. 또한 실험용 실제 하드웨어와 추출한 파라메터로 생성한 가상 하드웨어에 대하여 API를 이용하여 구현 가능성 및 지연 시간 요소를 분석하였다. 파라메터를 설정하여 실험 결과, VDD로 구성된 가상 하드웨어의 기능이 실제 하드웨어의 기능을 대치할 수 있었으며, 임의의 하드웨어에 대해서도 VDD를 이용한 개발 모형을 적용할 수 있음을 입증하였다. 파라메터의 변경으로 하드웨어의 기능을 실시간으로 변경시키면서 충분한 실험과 검증을 거쳐 하드웨어를 최적화 할 수 있다.

표 2는 실험용으로 제작한 실제 하드웨어와 이에

표 2. 실험용 실제 하드웨어 및 가상 하드웨어에 대한 실험 결과(단위 : sec)
Table 2. The experimental result on real testing hardware and the virtual hardware(unit : sec)

함 수 명	실제 하드웨어	가상 하드웨어	실제 하드웨어의 지연요소	비 고
CardStatus( )	1.2211	1.2211	하드웨어의 입출력	카드정보 유지
CardAuthentication( )	1.2211	1.2211	하드웨어의 입출력	카드정보 유지
ReadCardAttribute( )	0.0412	0.0412	하드웨어의 입출력	카드정보 유지
ChangePassword( )	0.0412	0.0412	사용자 입력	패스워드 유지
RegisterMainUserId( )	0.0412	0.0412	사용자 입력	카드정보 유지
DeleteMainUserId( )	0.0412	0.0412	사용자 입력	카드정보 유지
RegisterMinorUserId( )	0.0412	0.0412	사용자 입력	카드정보 유지
DeleteMinorUserId( )	0.0412	0.0412	사용자 입력	카드정보 유지
ReadRegisteredUserId( )	0.0412	0.0412	실행지연	등록리스트 유지
ReadLogData( )	0.0412	0.0412	하드웨어의 입출력	이력 유지
ReadTime( )	0.0034	0.0034	내부 시계칩 읽기	시스템 시간
ReadLibraryVersion( )	0.0001	0.0001	메모리 읽기	버전 유지
DiagnoseDevice( )	2.0023	2.0023	하드웨어의 입출력, 실행지연	S/W 진단
RequestUserAthenData( )	0.0412	0.0412	하드웨어의 입출력	약속
GenerateUserAthenData( )	0.0412	0.0412	하드웨어의 입출력	약속
VerifyUserAthenData( )	0.0412	0.0412	하드웨어의 입출력	약속
StartBufferedDataProcessing( )	0.0608	0.0608	초기화 지연시간	초기화
ExecuteBufferedDataProcessing( )	92.967	92.967	실행지연	처리 시간
StopBufferedDataProcessing( )	0.0006	0.0006	종료 지연시간	무시가능
ProcessDiskFileData( )	33.196	33.196	실행, 파일입출력지연	1 MB
ProcessMemoryFileData( )	32.833	32.833	실행, 파일입출력지연	1 MB

대한 가상 하드웨어에 대하여 VDD 개발 모형을 통해 실험한 결과이다. 실행 시간은 어플리케이션 바로 밑단에서 측정하였으며 결과치는 소수점이하 4자리에서 반올림하여 기록하였다. 실제 응용에서 구현되는 방식과 운용되는 컴퓨터 시스템에 따라 다를 수 있으므로 절대적인 수치는 아님을 밝혀 두며, 본 논문에서 사용한 전산기의 운영체계는 Windows NT workstation이며, Pentium 166MHz의 CPU와 32Mbyte의 메모리로 구성된 환경에서 실험하였다.

가상 하드웨어를 이용한 개발 모형을 통해 여러 형태의 하드웨어에 대한 기능을 파라메터를 변경하면서 어플리케이션에 적용하여 실험하고, 실험 결과에 대하여 충분한 검증을 통하여 어플리케이션에 대한 최적의 하드웨어를 도출할 수 있다. 가상 하드웨어를 이용하여 어플리케이션을 작성할 수 있어 하드웨어의 개발에 관계없이 독립적이면서 병렬 적으로 프로그램을 개발할 수 있어, 전체적인 개발 기간의 단축 효과를 얻을 수 있다.

V. 결 론

본 논문에서는 응용 체계의 구현에 소요되는 시간을 단축시키고, 개발 단계에서 완성될 응용 체계에 대한 성능을 보장하며, 요구 규격의 변화에 능동적으로 대처할 수 있는 가상 하드웨어를 이용한 개발 방법을 제안하였다. 하드웨어를 대신하는 가상 하드웨어는 소프트웨어 모듈이며, 파라메터를 설정하여 하드웨어의 기능을 수행한다. 하드웨어의 규격 변경은 파라메터의 변경으로 가능하여 실시간으로 응용 체계와 연동하여 성능을 평가할 수 있다. 제안한 개발 모형에 대한 적용 가능성의 입증을 위해, 실험용으로 제작한 하드웨어와 하드웨어로부터 파라메터를 추출하여 생성한 가상 하드웨어를 응용 체계에 적용하여 실험 결과를 얻었다. 실험 결과를 분석한 결과, 하드웨어를 이용한 결과와 가상 하드웨어를 이용한 결과가 동일하여 가상 하드웨어로 하드웨어를 대체할 수 있음을 보였다.

이로써, 기존의 응용 체계에 보안 서비스 등 특별한 기능을 추가적으로 제공해야하는 상황에서 유용하게 응용될 수 있으며, 응용 체계의 운용 환경에 최적의 하드웨어 규격을 실제 제작하기 이전에 알 수 있기 때문에 적용 가능성은 충분하다고 사료된다.

향후 연구 방향은 다양한 하드웨어에 대하여 일반적이며 보다 실질적인 모델링으로 발전시켜, 급속도로 발전하는 다양한 어플리케이션에 능동적으로 각종 서비스를 적용할 수 있도록 한다.

참 고 문 헌

1. Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall International Editions, 1992.
2. Art Baker, *The Windows NT Device Driver Book A Guide For Programmers*, 1997.
3. BlueWater System, Inc, *WinDK DDK Class Library-Users Manual*, 1996.
4. J. Linn, *Generic Security Service Application Program Interface*, RFC 1508, 1993.
5. Jeff Prosise, *Programming Windows 95 with MFC*, 1996.
6. Jeffrey Richter, *Advanced Windows NT*, Microsoft Press, 1995.
7. Mark Andrews, *Visual C++ 4.x Windows NT 4.0 Programming*, 1997.
8. Microsoft, *Microsoft Visual C++ MFC Monthly Technical Seminar*, Oct, 1996.
9. MSDN, *Win32 SDK(for Windows 95 and Windows NT Workstation 4.0)*, July 1996.
10. NIST, *Federal Information Processing Standards Publication Series(FIPS PUB XXX)*, May 1994.
11. NSA, *Fortezza Cryptologic Interface Programmers Guide, Revision 1.52*, Jan 1996.

고 재 영(Jae Young Koh)정회원

1984년 2월 : 전북대학교 전자공학과(공학사)

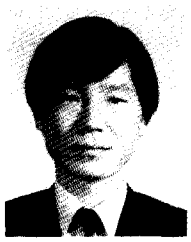
1992년 8월 : 전북대학교 대학원 전자공학과(공학석사)

1993년 3월~현재 : 전북대학교 전자공학과 박사과정

1984년 3월~현재 : 국방과학연구소 재직

※주관심분야 : 컴퓨터 네트워크, 전송부호화, 정보보호, 데이터 통신

김 봉 환(Bong Hwan Kim)정회원
1992년 2월 : 충남대학교 전산과학과(이학사)
1994년 2월 : 충남대학교 대학원 전산과학과(이학석사)
1994년 3월~현재 : 국방과학연구소 재직
※주관심분야 : 분산 시스템, 고성능 컴퓨팅, 정보보호



송 상 섭(Sang Seob Song) 정회원
1978년 2월 : 전북대학교 전기공학과(공학사)
1980년 2월 : 한국과학기술원 전기 및 전자공학과(공학석사)
1990년 8월 : 캐나다 마니토바대학 전기 및 컴퓨터공학과(공학박사)
1981년 4월~현재 : 전북대학교 전기·전자·제어공학부 교수
※주관심분야 : 채널부호이론, 고속전송기술, ATM ASIC, 무선 멀티미디어 전송 기술