이중 커널 구조의 OS를 위한 IEEE1394 디바이스 드라이버의 설계 및 구현

(Design and Implementation of IEEE1394 Device Driver for Dual Kernel OS)

정 기 훈 † 오 주 용 †† 강 순 주 †††
(Gi-Hoon Jung) (Ju-Yong Oh) (Soon-Ju Kang)

요 약 본 논문에서는 이중 커널 OS인 RTLinux에서 실시간, 비실시간 커널의 응용 프로그램을 동시에 지원하기 위한 IEEE1394 디바이스 드라이버의 구조를 설계 및 구현하였다. 제안한 이중 커널 OS를 위한 디바이스 드라이버는 양 커널의 태스크를 동시에 지원할 수 있는 장점을 가진다. 이와 더불어 제안된 디바이스 드라이버는 실시간 커널측의 작업 요청을 우선적으로 처리하도록 구성하여 실시간성 보장이 가능하도록 배려하였다. 이 디바이스 드라이버의 구조는 RTLinux뿐만 아니라 이중 커널 시스템을 위한 디바이스 드라이버 설계에 도움이 될 것이다.

키워드 : IEEE1394, 디바이스 드라이버, RTLinux, 소프트웨어 구조, 이중 커널

Abstract In this paper, we propose an architecture of IEEE1394 device driver for RTLinux. The device driver has two interfaces for applications running on the RTLinux kernel and Linux kernel. With the interfaces, the device driver simultaneously supports RT-Thread of RTLinux kernel and user level process of Linux kernel. This architecture could be a reference for designing other device driver on the dual kernel platform.

Key words : IEEE1394, Device Driver, RTLinux, Software Architecture, Dual Kernel

1. 서 론

고속 시리얼 버스인 IEEE1394[1]는 비동기 전송(Asynchronous) 방식과 등시성 전송(Isochronous) 방식을 지원한다[2]. 비동기 전송은 요청/응답(Request/Response) 방식으로 비실시간 서비스에서도 사용 가능한 반면, 등시성 전송은 125us의 일정한 주기로 브로드캐스트 방식으로 전송하므로 실시간 특성을 가진다. 따라서 등시성 전송을 이용하는 서비스는 125us라는 주기를 맞추기 위해 실시간 특성을 가질 필요가 있다.

실시간 OS인 RTLinux[3]는 이중 커널 구조를 가지고 있다. 비실시간 Linux 커널과 실시간 RTLinux 커널이 공존하며, Linux의 비실시간 응용 프로그램과 RTLinux의 실시간 응용 프로그램을 동시에 수행할 수 있고, 둘 간의 연동 작업도 가능한 OS이다. 이러한 RTLinux를 기반으로 하는 디바이스 드라이버는 두 커널의 응용 프로그램 모두에 서비스를 제공할 수 있어야 한다. 그러나 현재 RTLinux 상에서 구현된 디바이스 드라이버의 개발 사례들을 살펴보면 RTLinux RT-Thread[4]와 Linux process를 동시에 지원하는 디바이스 드라이버는 매우 적고, 구조가 체계화되어 있지 않다. 공식적으로 발표된 자료로 보면 Time-Triggered Ethernet Protocol을 채용한 디바이스 드라이버[5]가 RTLinux상에서 듀얼 커널 지원을 고려한 유일한 연구 사례이다.

따라서 본 논문에서는 RTLinux의 실시간 스레드와 Linux의 비실시간 태스크를 동시에 지원하기 위한 디바이스 드라이버의 구조를 RTLinux용 IEEE1394 디바이스 드라이버의 설계 및 구현에 적용하였다. 이러한 구조는 비실시간 특성의 비동기 전송과 실시간 특성의 등시성 전송 방식을 가진 IEEE1394용 응용 프로그램 개발에 적합한 장점을 가지고 있다.

· 본 연구는 한국과학재단 특정기초연구(R01-2003-000-10252-0)지원으로 수행되었음
† 정 회 원 : 경북대학교 디지털기술연구소 연구원
jghlof@palgong.knu.ac.kr
†† 비 회 원 : 경북대학교 전자공학과
anyong@palgong.knu.ac.kr
††† 정 회 원 : 경북대학교 전자전기컴퓨터학부 교수
sjkang@ee.knu.ac.kr
논문접수 : 2004년 5월 15일
심사완료 : 2004년 11월 25일

본 논문은 2장에서 기본 개념 및 배경, RTLinux의 특징과 실시간-비실시간 태스크 동시지원을 위한 디바이스 드라이버의 설계 요건에 대해 설명한다. 3장에서 제안된 디바이스 드라이버의 소프트웨어 구조를 살펴보고, 4장에서 성능 평가 결과를 보이고 5장에서 결론을 맺는다.

2. 기본 개념 및 배경

2.1 IEEE1394 규격 상에서 디바이스 드라이버의 기능적인 위치

IEEE1394 규격에서는 크게 4개의 레이어로 IEEE 1394를 설명한다. 이에 따르면 IEEE1394는 Physical Layer, Link Layer, Transaction Layer, Bus Management Layer로 구성된다. 이러한 기능 레이어들은 대부분 하드웨어적으로 구현되어 있다.

일반적으로 IEEE1394 디바이스 드라이버를 작성할 때는 Bus Management Layer와 Transaction Layer 기능의 일부를 소프트웨어에서 분담하도록 구성하게 된다. 그러나 1394 전송 체계 전체를 기준으로 보면 매우 미미한 수준의 기능을 분담하기 때문에, IEEE1394 디바이스 드라이버는 4개 레이어들의 상부에 위치하면서 기능을 제어 및 관리하는 것으로 볼 수 있다.

2.2 제안된 디바이스 드라이버와 일반 Linux용 디바이스 드라이버의 차이점

일반 Linux 커널의 경우, 커널 레벨에서는 오직 하나의 태스크만이 동작하게 된다. 이 커널 태스크는 모든 자원을 점유하고 운영하며, 디바이스 드라이버는 커널 태스크와 분리되어 동작할 수 없다. 디바이스 드라이버는 커널 태스크의 일부로 운영되며, 독자적인 태스크를 가지지 못한다. 이러한 특징에 따라 일반 Linux용 디바이스 드라이버는 커널에 함수를 등록하여 운영하는 형식을 갖추고 있다. 즉, 디바이스 드라이버는 서비스에 필요한 다수의 콜백 함수들을 커널에 등록해서 운영하므로, 유저 측이 커널에 요청할 때에만 디바이스 드라이버의 기능이 동작하게 된다. 따라서 능동적인 정보 수집 및 응용 프로그램 지원 작업이 불가능하다[6].

반면 RTLinux의 경우 커널에서 멀티스레드 작업이 가능하다. RTLinux의 커널 레벨에서 운영되는 스레드를 RT-Thread라고 하는데, RTLinux는 모든 RT-Thread를 차별하지 않으므로 RTLinux의 커널 레벨에서 운영되는 디바이스 드라이버와 실시간 어플리케이션은 구분하여 관리하지 않는다. 따라서 RTLinux에서 디바이스 드라이버와 어플리케이션을 구분하는 기준은 개발자의 제작 의도에 따라 결정된다. 이러한 RTLinux 상에서 개발되는 디바이스 드라이버는 특정 하드웨어 자원을 점유하고, 그 자원을 커널 안의 다른 RT-Thread에 제공하는 서비스를 하게 된다. 이 경우 해당 하드웨어 자원을 RTLinux에서 점유하기 때문에 일반 Linux 커널의 디바이스 드라이버는 운영이 불가능하다. 따라서 유저 레벨의 일반 Linux용 어플리케이션은 해당 자원을 사용할 수 없게 된다. 역으로 일반 Linux용 디바이스 드라이버가 특정 자원을 점유하면, 해당 자원을 커널 스레드에서는 사용할 수 없게 된다. 즉, 디바이스 드라이버가 하나의 커널을 위한 인터페이스만 준비하고 있다면, 동시 지원은 불가능한 것이다

제안된 디바이스 드라이버는 RTLinux의 커널 스레드를 위한 인터페이스를 제공하면서, 유저 레벨의 일반 Linux용 어플리케이션을 위한 인터페이스도 준비하여 이중 커널을 동시 지원하고 있다. 또한, RTLinux의 실시간성 보장을 저해하지 않기 위해, 일반 Linux측의 인터페이스는 우선순위를 최하로 설정하여, RTLinux의 실시간성 보장 가능성을 높였다.

2.3 실시간-비실시간 커널 동시지원을 위한 디바이스 드라이버의 설계 요건

RTLinux는 경성 실시간(Hard Real Time) 프로그램을 운영할 수 있으며, 비실시간 Linux 커널과 실시간 RTLinux 커널을 동시에 운영하므로 범용 OS와 실시간 OS의 특징을 모두 가진다. 이러한 이중 커널 환경을 위한 디바이스 드라이버는 두 커널을 동시에 지원할 수 있어야 하므로 일반적인 디바이스 드라이버와는 다른 형태의 구조를 필요로 한다. 특히 두 커널을 위한 인터페이스를 모두 갖추고 있어야 할 것이다.

또한 실시간-비실시간 커널을 동시에 지원하는 디바이스 드라이버는 이중 인터페이스 이외에도 실시간성 보장측면을 최우선으로 고려해야 한다. 디바이스 드라이버의 문제로 인해 실시간 태스크가 데드라인을 만족시키지 못한다면 동시지원의 의미를 잃게 된다. RTLinux의 실시간 커널은 비실시간 커널에 대해서 선점권을 가지고 동작하기 때문에 공유 자원에 대한 동기화 문제를 피하면 실시간성을 보장할 수 있다. RTLinux는 커널의 기능을 매우 간결하게 축소하고, 복잡한 스케줄링 문제들은 응용 프로그램에서 해결하는 것을 원칙으로 하고 있다. 일반적으로 디바이스 드라이버도 커널의 운영원칙을 따르게 된다.

RTLinux 환경에서 동작하는 디바이스 드라이버는 다수의 실시간 커널 RT-Thread 및 일반 Linux의 비실시간 프로그램이 동시에 서비스 요청을 할 수 있으므로 동기화로 인한 실시간성 문제를 고려해야 한다. 또한 디바이스 드라이버 모듈도 하나의 실시간 태스크로 운영되므로, 서비스를 요청하는 작업들과 디바이스 드라이버 사이의 우선순위에 대해서도 고려해야 한다. 특히 RTLinux내의 RT-Thread 간에는 비선점형 스케쥴링

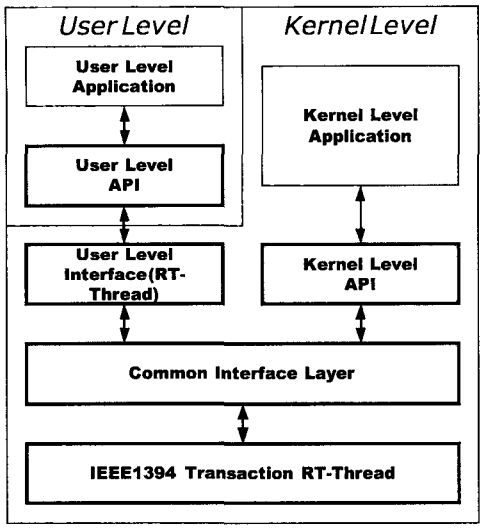

그림 1 RTLinux용 IEEE1394 디바이스 드라이버의 계층도

방식이 적용되므로 동기화 문제를 피하기 위해, 디바이스 드라이버의 우선순위를 최고로 설정해야 한다. 즉, 디바이스 드라이버가 작업 중일 때는 타 RT-Thread의 방해를 받지 않도록 하여야 한다. 이 경우 해당 디바이스 드라이버에 작업을 요청한 실시간 태스크는 디바이스 드라이버에서 소요되는 작업 시간의 변동폭이 예측 가능한 범위로 유지되기 때문에 디바이스 드라이버로 인해 실시간성 보장이 안 될 가능성이 줄어들게 된다.

3. 제안된 디바이스 드라이버의 구조

3.1 제안된 디바이스 드라이버의 전체 구조

제안된 RTLinux를 위한 디바이스 드라이버는 유저 레벨의 비실시간 응용 프로그램과 커널 레벨 RT-Thread의 요청을 동시에 지원하게 된다. 이를 위해 디바이스 드라이버의 구조를 크게 다섯 부분으로 나누었다.

그림 1처럼 제안된 디바이스 드라이버는 최하에서부터 IEEE1394 트랜잭션 RT-Thread 레이어와 공동 인터페이스 레이어, 유저 레벨 응용 프로그램을 위한 인터페이스 레이어 및 유저 레벨 API와 커널 레벨 API로 구성되어 있다. 최하부의 IEEE1394 트랜잭션 RT-Thread 레이어는 실제 전송 트랜잭션을 수행하여 그 결과를 공동 인터페이스 레이어로 전달한다. 공동 인터페이스 레이어는 일반적인 트랜잭션 인터페이스 관련 함수들이 제공되어 상부 레이어에서 호출하여 이용할 수 있다.

공동 인터페이스 레이어 위에는 유저 레벨 인터페이스 레이어와 커널 레벨 응용프로그램을 위한 API로 구분된다. 커널 레벨 API는 디바이스 드라이버 내에서 커널 공유 함수 형태로 제공되며, 유저 레벨 인터페이스는 RT-FIFO를 이용하여 유저 레벨 API와 통신을 한다. 유저 레벨 인터페이스 레이어에서는 트랜잭션의 특성에 따라 유저 레벨 프로세스의 요청을 받아 대행하는 Agent RT-Thread가 내부적으로 운영된다. 마지막으로 유저 레벨 API는 유저 레벨 응용 프로그램의 일부로 수행되도록 라이브러리 파일로 제공된다. 이러한 듀얼

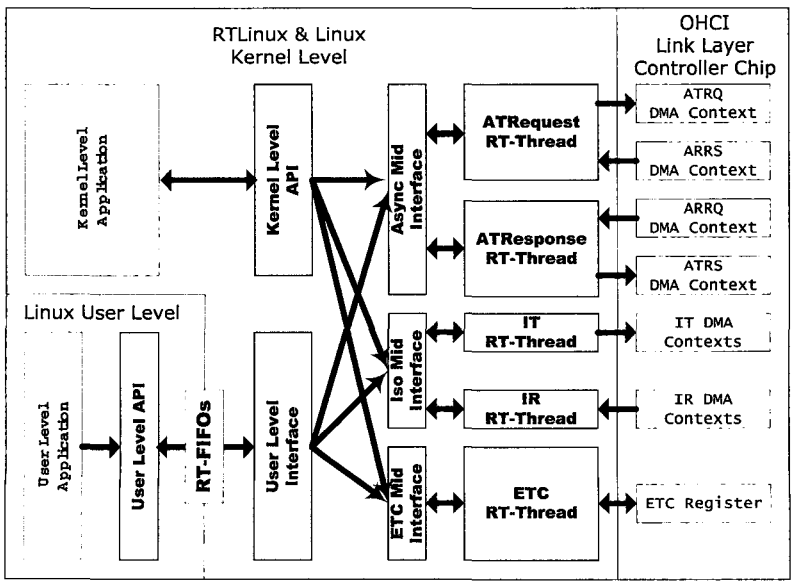

그림 2 서비스별로 분류한 디바이스 드라이버 구조도

커널 동시 지원 구조는 IEEE1394의 트랜잭션에 맞춰 좀 더 세밀하게 구현되었고, 실제 드라이버의 전체 형태는 그림 2와 같이 구성되어 있다.

디바이스 드라이버의 핵심은 전송 작업을 실제로 하는 5개의 하부 RT-Thread이다. 이것들이 IEEE1394의 비동기 요청, 응답, 등시성 송신, 수신과 기타 작업을 처리하게 된다. 커널 레벨 API 함수들은 디바이스 드라이버 내에 존재하며, RTLinux 커널 내의 모든 모듈이 접근 가능하도록 커널 공유 함수로 선언되어 있다. 유저 레벨 어플리케이션과의 인터페이스에는 10여개의 RT-FIFO를 사용하고 있다. 커널 레벨 응용 프로그램은 커널 API를 통해 작업을 요청하며, 우선순위도 유저 레벨에 비해 높게 배정을 받으므로 실시간성 보장이 가능하다. 이에 비해 유저 레벨 응용 프로그램은 우선순위가 최하이며, RT-FIFO를 거쳐서 요청하게 되므로 작업 효율이 떨어지게 된다.

디바이스 드라이버 내의 트랜잭션용 RT-Thread들은 우선순위가 디바이스 드라이버를 이용하는 응용 프로그램 RT-Thread들보다 높게 설정되어 있다. 이 설정은 서비스 요청을 받았을 때 최대한 빠른 시간 내에 작업을 완료하여 결과를 돌려주면서, 동시에 교착 상태를 미연에 방지하기 위함이다. 또한 디바이스 드라이버 내의 5개 RT-Thread들 간에도 우선 순위에 차등을 두었다. 높은 우선 순위를 기준으로 기타 작업, 등시성 송신, 수신, 비동기 응답, 요청의 순으로 우선 순위를 주었다. 이것은 IEEE1394 전송 특성을 고려한 순위이다.

3.2 비동기 전송 요청측의 구조

그림 3에서 비동기 전송의 요청측 응용 프로그램은 전송할 데이타를 준비한 뒤에 작업 요청을 하게 된다. 커널 레벨 응용 프로그램은 커널 레벨 API를 호출하여 직접적으로 전송 트랜잭션을 처리하는 RT-Thread에 요청을 하게 되며(1), 유저 레벨 응용 프로그램은 RT-FIFO를 통해서 간접적으로 User Agent RT- Thread를 통해서 요청하게 된다(2). 커널과 유저 레벨이 동시에 요청하게 되면 커널 레벨 쪽의 작업이 우선적으로 수행되게 되며, 하나의 작업이 완료되기 전에는 다른 작업이 수행될 수 없다. 한편, User Agent RT- Thread는 비실시간 작업을 전담하므로 우선순위가 최하로 설정되어 있다.

요청을 받은 전송 RT-Thread는 DMA 버퍼에 요청 패킷을 복사하고(3), 필요에 따라 GUID to Node ID Mapping Table[7]이나 Speed Map의 정보를 참고하여

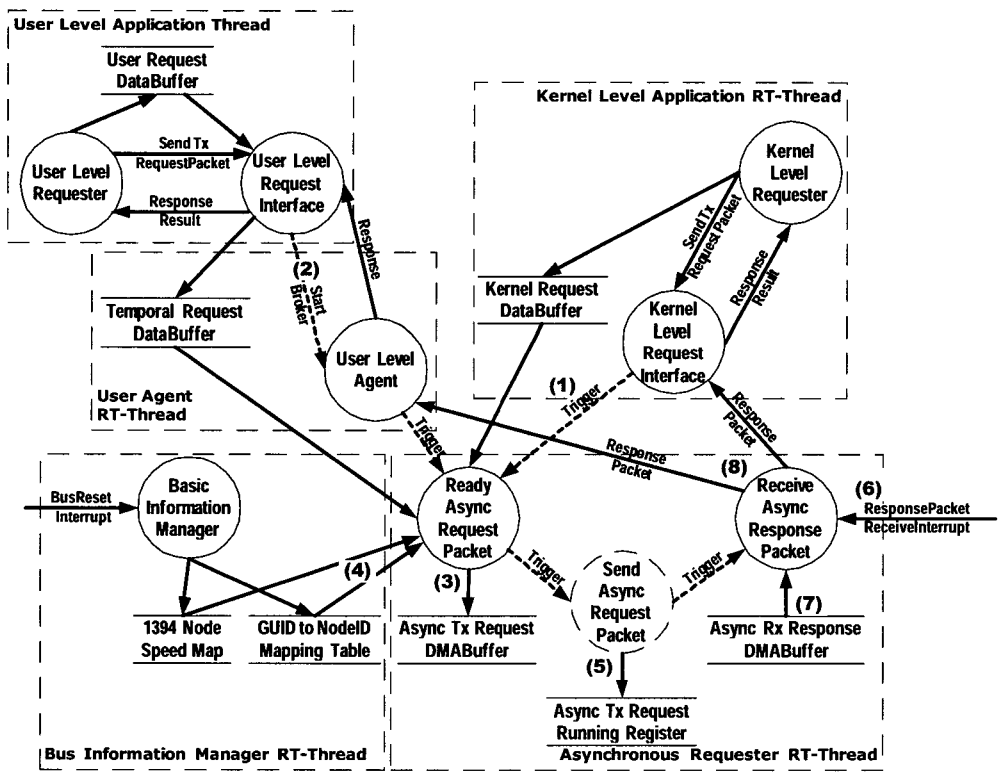

그림 3 비동기 전송 요청측의 디바이스 드라이버 자료 흐름도

목적 노드나 전송 속도에 관한 정보를 일부 수정한다(4). 전송 준비가 완료되면 DMA 레지스터를 제어해 전송한다(5). 이 과정 중에 사용하는 GUID to Node ID Mapping Table 서비스는 IEEE1394의 유동적인 버스 설정 변경 상황 속에서도 일정한 노드를 억세스하기 쉽도록 제공되는 서비스이며, Speed Map은 IEEE1394 망 내의 모든 노드 간의 전송 속도 정보가 기록되어 있다.

요청 패킷이 무사히 전송되면 상대측 노드에서 ack 패킷이 약 1μsec 정도 이내에 돌아오고, 좀 더 후에 응답 패킷이 오게 된다. 응답 패킷이 올 때까지 요청측은 대기한다. 응답 패킷을 수신하면 하드웨어에서 인터럽트가 발생하게 되며(6), 인터럽트를 받으면 응답 패킷 수신용 DMA 버퍼를 확인하여 패킷을 추출하고(7) 이를 요청해왔던 스레드에 돌려준다(8). 한편 전송 시에 참고하는 GUID 테이블과 Speed Map은 버스 리셋이 발생한 뒤에 모든 노드의 정보를 취합하여 갱신된다. 항상 최근의 정보를 유지해주기 때문에 응용 프로그램에서 각 노드의 고유한 GUID를 이용해 전송 요청을 하면, 항상 특정 노드로 전송이 가능하다. 비동기 전송 요청측은 한 번에 하나의 요청 패킷을 보내고 응답 패킷이 오거나 에러가 발생할 때까지 기다리도록 구현되어 있다. 1394 규격에서 비동기 전송의 요청과 응답은 항상 1:1로 주고받도록 규정하므로 이를 따라서 구현되어 있는 것이다.

3.3 비동기 전송 응답측의 구조

그림 4에서 요청 패킷은 대상 노드에 특정 1394 어드레스의 내용을 읽거나 쓰는 작업을 지시하게 된다. 이에 대응하기 위해 비동기 전송 응답측에서는 사전에 특정 1394 어드레스 범위를 대응하는 콜백 함수를 등록하게 되어 있다. 응답측의 응용 프로그램은 대응하여 처리할 1394 어드레스 범위를 사전에 디바이스 드라이버에 등록하게 된다(1). 이 후 요청 패킷을 수신하면 하드웨어 인터럽트가 발생하며(2), 수신한 요청 패킷을 디바이스 드라이버가 분석하게 된다(3). 요청 패킷이 접근할 어드레스가 콜백 함수 리스트에 등록되어 있다면(4), 등록된 콜백 함수가 호출된다. 호출된 콜백 함수는 등록된 응용 프로그램의 일부이지만, 디바이스 드라이버의 태스크로써 동작하게 된다. 콜백 함수는 응답 패킷에 적재될 데이타나 결과값을 디바이스 드라이버에게 돌려주게 된다(5). 디바이스 드라이버는 콜백 함수의 결과값과 요청 패킷의 정보를 조합하여 응답 패킷을 준비한다. 완성된 응답 패킷은 DMA Buffer에 적재하게 된다(6). 이후 DMA 레지스터를 제어하여 응답 패킷을 전송하고, ack를 수신하면 응답 패킷 전송이 완료된다(7). 만약 응답측 프로그램이 서비스를 종료하고자 한다면, 등록된 콜백 함수 정보를 삭제하게 된다(8).

이와 같이, 응답측의 로컬 호스트 메모리의 어드레스와 1394 어드레스 간의 매핑 서비스를 위해서 콜백(Callback) 함수를 이용하였다. 방대한 1394 어드레스

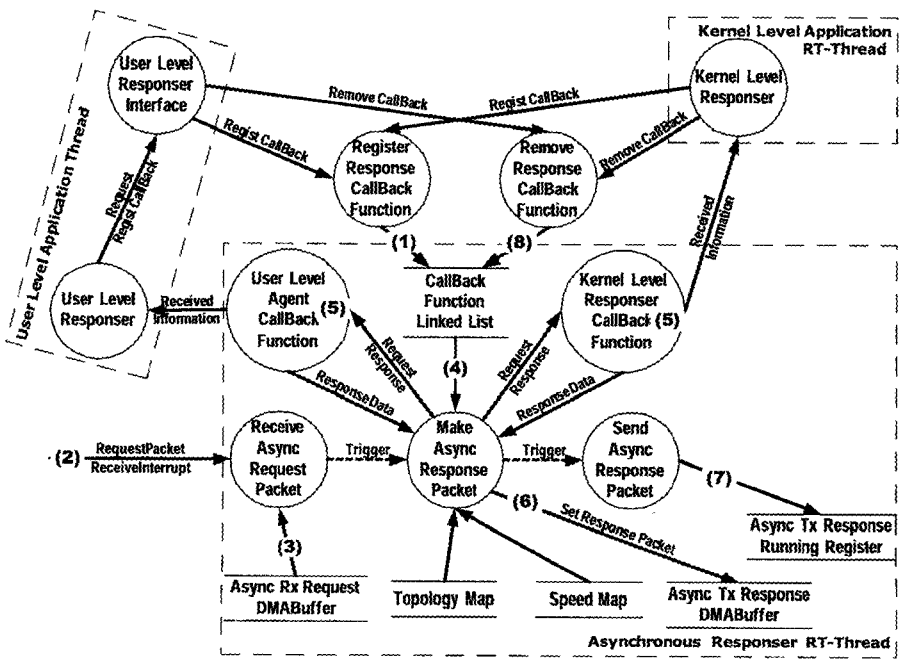

그림 4 비동기 전송 응답측의 디바이스 드라이버 자료 흐름도

범위를 고려하여 디바이스 드라이버의 콜백 함수 등록 레지스트리는 링크드 리스트로 구현하였다. 등록 레지스트리는 OHCI 규격[8]을 따라, Middle Address, Upper Address, CSR Address 영역[9]으로 나누어서 관리한다.

비동기 전송 응답측의 작업은 하나의 RT-Thread를 통해 처리하게 되며, 응답 패킷을 보낼 때까지 다른 패킷을 처리하지 못하도록 구성하였다. 비동기 전송은 패킷 단위로 볼 때 장시간의 작업을 필요로 하는 경우가 거의 없으므로, 최대한 응답을 빨리 할 수 있도록 간결하게 구성한 것이다. 이렇게 콜백 함수를 이용하므로 디바이스 드라이버는 구조적으로 간결해지고, 응답측의 처리방식을 유동적으로 변경할 수 있으므로 효율적인 프로그램 구현이 가능하다.

4. 성능측정 및 검증

4.1 검증 환경

디바이스 드라이버의 비동기 전송 성능 측정을 위해 세가지 과정을 거쳤다. 우선 실시간 프로그램과 비실시간 프로그램을 작성하여 각각을 독립 수행할 때 소요시간을 측정하였다. 또한 네덜란드 dap design사[10]의 FireSpy400 네트웍 어널라이저를 통해 실제 전송 상황을 확인하였다. 마지막으로 두 프로그램을 각각 하나씩 동시에 수행하는 상태에서 성능 측정을 하였다. 전송 소요 시간은 응용 프로그램에서 작업 요청 시 응답 패킷을 받는데 걸리는 시간을 측정하였다. 패킷 당 데이타 크기는 4~2048 바이트까지 변경하면서, 읽기와 쓰기 작업을 각각 측정하였다. 검증을 위한 네트웍은 그림 5와 같이 구성되었다.

4.2 비동기 전송 성능 측정

4.2.1 단독 비동기 전송 수행의 성능

읽기와 쓰기 전송을 각각 10000회씩 반복하고 그 결과에서 평균과 표준 편차를 계산하였다. 읽기 작업은 상대편의 디바이스 드라이버가 관리하는 Topology Map 정보를 읽어오도록 하였고, 쓰기 작업은 FCP 영역[11]에 접근하도록 구현하였다.

그림 6의 그래프는 패킷 당 전송 데이타 크기에 따른 전송 소요 시간을 보여 주고 있다. 제일 느린 성능을 보여준 것은 유저 레벨 응용 프로그램이 쓰기 요청을 했을 경우이며, 가장 빠른 성능을 보여준 것은 커널 레벨 응용 프로그램이 읽기 요청을 했을 경우이다. 가장 빠른 커널 레벨 읽기 요청의 경우 2048바이트 전송에 200μsec이하의 시간 밖에 걸리지 않는 높은 전송 성능을 보여주고 있다. 전송 대역폭으로 보면 4바이트 전송에 56.6KB (452.8Kb)/sec, 2048바이트 전송 시 11.2MB (89.6Mb)/sec의 전송량이며, 실제 전송 시 패킷 하나당 16바이트의 헤더와 8바이트의 CRC 데이타가 추가되는 점을 고려하면 순수한 전송 성능은 이보다 높은 것으로 예상할 수 있다. 표 1은 그림 6의 결과에 대한 표준 편차 표이다. 유저 레벨에서 작업을 요청한 경우가 커널 레벨에서 작업을 요청한 경우보다 약 1μsec 정도 더 변동이 있는 것으로 나타나고 있다.

그림 7은 커널 레벨 응용 프로그램에서 읽기, 쓰기의 성능을 측정할 때 네트웍 어널라이저를 이용해 포착한

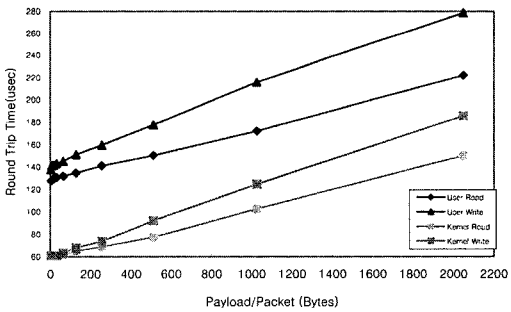

그림 6 단독 수행시 비동기 전송의 소요 시간 측정

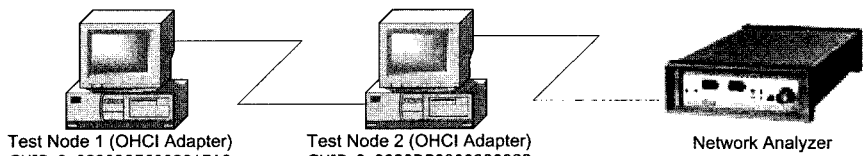

그림 5 성능 평가를 위한 시스템 구성

표 1 단독 수행시 비동기 전송 소요 시간의 표준 편차(μsec)

Payload/Packet (Bytes)	4	8	16	32	64	128	256	512	1024	2048	평균
User Level Read	4.42	5.43	5.80	6.35	5.41	**4.24**	**7.37**	6.87	4.84	6.28	**5.70**
User Level Write	6.50	**3.78**	4.65	5.41	5.84	**7.46**	5.15	6.07	5.14	5.68	**5.57**
Kernel Level Read	**6.38**	5.89	5.78	4.80	4.38	5.37	4.68	5.03	**3.21**	4.36	**4.99**
Kernel Level Write	5.51	5.45	5.37	4.20	5.06	5.45	**3.26**	**6.00**	3.40	5.08	**4.88**

표 2 실시간-비실시간 동시 수행시의 성능 표준 편차

Payload/Packet(Bytes)	4	8	16	32	64	128	256	512	1024	2048	평균
Kernel Read 단독 수행	6.38	5.89	5.78	4.80	4.38	5.37	4.68	5.03	3.21	4.36	4.99
Kernel Read 동시 수행	6.72	6.74	6.63	7.29	7.13	5.37	3.10	5.94	4.61	5.50	5.90
User Read 단독 수행	4.42	5.43	5.80	6.35	5.41	4.24	7.37	6.87	4.84	6.28	5.70
User Read 동시 수행	36.01	36.51	36.38	37.11	37.52	38.04	39.08	44.24	52.67	75.72	43.33

실제 패킷의 전송 상황이다. 4, 2048바이트의 데이타를 읽기와 쓰기 전송을 할 때의 상황이 그림에서 나타나고 있다. 각각의 왼쪽 상단에 보이는 시간이 하나의 요청 트랜잭션이 시작될 때부터 끝날 때까지의 시간으로 그림에서는 화살표로 나타난 구간 사이의 시간이다. 소요 시간은 그림 6에서 보인 측정 결과에 가깝게 나타난다. 측정 구간 내에 있는 연한 색의 패킷은 요청에 대한 응답 패킷이며 작은 막대로 표시된 패킷은 Cycle Start 패킷으로 등시성 전송에 관계된 패킷이다.

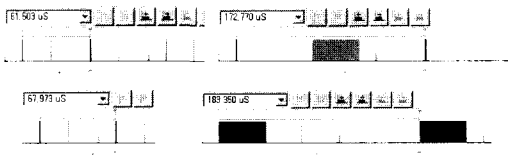

그림 7 네트웍 어널라이저로 본 비동기 전송 상황

4.2.2 실시간-비실시간 작업 동시 수행의 성능

RT-Thread로 실시간 작업 하나를 실행하면서 유저 레벨 비실시간 작업 하나를 동시에 수행하여 전송에 소요되는 시간을 측정하였다. 실시간 작업이 비 실시간 작업보다 훨씬 빨리 처리되는 점을 감안하여 최대한 동일한 시간 동안 수행되도록 고려했으며, 각각의 평균과 표준 편차를 계산해서 단독 수행시의 결과와 비교했다.

그림 8에서 나타나듯 실시간 작업은 단독 수행 때와 별차이가 없는 성능을 나타냈으나 비실시간 작업은 평균 27μsec 정도의 성능 저하가 나타나고, 표 2에서 나타나듯 표준편차는 38μsec 정도 늘어나는 현상이 나타났다. 이는 유저 레벨에서 수행되는 비실시간 작업이 커널 레벨에서 수행되는 실시간 작업에 밀려 대기하는 시간이 늘어났음을 보여주고 있는 것이다. 이러한 결과에서 볼 때 이중 커널 지원 디바이스 드라이버가 실시간성 보장에 지장을 주지 않으면서, 비 실시간 작업도 동시 지원이 가능함을 알 수 있다.

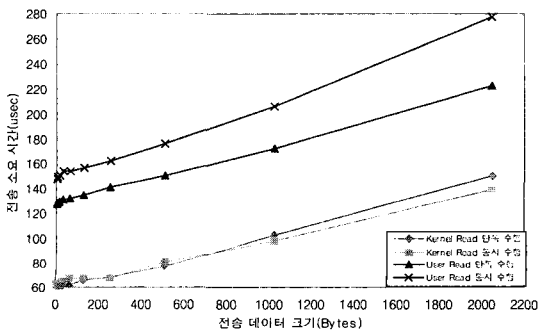

그림 8 실시간 비실시간 동시 수행시의 비동기 읽기 전송 성능 비교

5. 결 론

본 논문에서는 실시간 OS인 RTLinux에서 실시간-비실시간 동시지원을 위한 이중 커널 지원 디바이스 드라이버의 구조를 제안하고, IEEE1394 디바이스 드라이버의 구현에 적용하였다. 이를 통해 실시간 RT-Thread 태스크와 비실시간 유저 레벨 태스크를 동시에 지원하면서 실시간성을 보장하는 이중 커널 지원 디바이스 드라이버의 구현이 가능함을 보였다.

제안된 디바이스 드라이버에서는 이러한 동시 지원 구조를 구현하기 위해 유저 레벨과 커널 레벨 각각을 위한 인터페이스를 가지고 있으며, 커널 레벨의 실시간성 보장을 위해 유저 레벨 응용 프로그램을 최저 우선순위의 에이전트 RT-Thread를 통해 지원하고, 디바이스 드라이버의 우선순위를 최고로 설정하였다. 또한, 동기화 기법과 Linux 커널의 공유 함수, RT-FIFO를 사용하여 다수의 RT-Thread와 유저 어플리케이션이 서로를 의식할 필요 없이 디바이스 드라이버의 서비스를 이용할 수 있도록 하였다. 그리고 제안된 구조가 커널 레벨의 실시간성을 보장하면서 이중 커널 지원이 가능함을 실험을 통해 보였다.

이러한 실시간-비실시간 태스크 동시지원 구조를 사용하게 되면 유저 레벨 비실시간 프로세스의 범용성과 커널 레벨 RT-Thread의 실시간성 보장의 이점을 모두 활용한 응용 프로그램을 개발할 수 있다. 뿐만 아니라 이중 커널 시스템에서 이러한 디바이스 드라이버 구조가 좋은 방안으로 사용될 수 있을 것이다.

참 고 문 헌

[1] IEEE1394,Standard for High Performance Serial Bu, 1995.

[2] Don Anderson, *FireWire System Architectur*, 2nd Ed., MindShare,Inc., 1999.

[3] FSMLABS, http://fsmlabs.com/community.
[4] RTLinux Homepage,http://www.fsmlabs.com/products/rtlinuxpro/rtlinuxpro.htm.
[5] Stefan Lankes and Michael Reke, "A Time-Triggered Ethernet Protocol for Real-Time CORBA," *5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2002)*, Washington DC, USA, April 2002.
[6] Alessandro Rubini and Jonathan Corbet, *Linux Device Driver*, 2nd Ed., Reilly, June 2001.
[7] Joo-Yong Oh, Soon-Ju Kang, and Kyeong-Deok Moon, "Design and Implementation of CORBA ORB on top of IEEE1394-based Home Networ," *International Conference on Communications in Computing(CIC 2002)*, Las Vegas, USA, June 2002.
[8] 1394 Open Host Controller Interface Specification Release 1.1, 2000.
[9] IEEE Standard Control and Status Register (CSR) Architecture for Microcomputer Buses, July 1992.
[10] DapDesign, http://www.dapdesign.com
[11] IEC 61883-1, Consumer audio/video equipment Digital Interface, 1998-02.

정 기 훈

2001년 경북대학교 전자전기공학부 학사 2003년 경북대학교 전자공학과 석사 2003년~현재 경북대학교 디지털기술연구소 연구원. 2004년~현재 경북대학교 전자공학과 박사과정 재학 중. 관심분야는 Real-Time System, System Software, Home Network, IEEE1394

오 주 용

2000년 경북대학교 전자전기공학부 학사 2002년 경북대학교 전자공학과 석사 2002년~현재 경북대학교 전자공학과 박사과정 재학 중. 관심분야는 홈 네트웍 미들웨어, IEEE1394, real-time systems

강 순 주

1983년 경북대학교 전자공학과 학사 1985년 한국과학기술원 전자계산학과 석사. 1995년 한국과학기술원 전자계산학과 박사. 1985년~1996년 한국원자력연구소 연구원, 핵인공지능연구실 선임 연구원, 전산정보실 실장. 2000년 7월~2002년 8월, University of Pennsylvania, Dept. of Computer and Information Science, 객원 연구 교수. 1996년~현재 경북대학교 전자전기컴퓨터학부 정보통신전공 부교수. 관심분야는 Real-Time System, Software Engineering, Knowledge-Based System