논문 2010-05-04

차량용 MOST 네트워크를 위한 POSIX 기반의 Network Service 설계 및 구현

(Design and Implementation of MOST Network Service over POSIX)

이 무 열, 정 성 문, 진 현 욱*
(Mu-Youl Lee, Sung-Moon Chung, Hyun-Wook Jin)

Abstract: The automotive industry tries to provide infotainment systems to emerging automobiles. Since the infotainment systems require various peripheral devices and network connectivity, legacy operating systems such as Linux and Windows can be much preferred due to its plentiful device drivers and multimedia applications while the operating systems following OSEK standard are used for automotive electronic control units. Thus it is highly desired that the system software supporting infotainment applications can be portable over different legacy operating systems providing unified programming interfaces. The majority of legacy operating systems support POSIX interfaces for application development. MOST is an automotive network standard for infotainment systems. Network Service defines the protocol stacks for MOST control data, which is essential to implement infotainment applications over MOST. In this paper, we suggest a POSIX-based Network Service so that we can utilize legacy device drivers and applications for automotive infotainment systems. We measure the performance of the POSIX-based Network Service and show that its overhead is not significant.

Keywords : MOST, Network service, 차량용 인포테인먼트 시스템, POSIX, 리눅스

Ⅰ. 서론

현재 차량들은 차량의 구동에 관하여 계측 제어하는 수많은 전자 기기 및 부속들을 포함하고 있다. 이 장치들은 필요에 따라 차량용 네트워크인 CAN, LIN, FlexRay 등을 사용하여 기기간 통신을 하게 된다.

한편 현재 차량들은 차량의 운행에 관한 기능 이외의 네비게이션, TV, 비디오, 인터넷 서비스 등의 멀티미디어적인 기능을 요구하게 되었다. 이러한 시대적인 요구에 차량은 각종 인포테인먼트 시스템들을 장착하기 시작하였다. 하지만 기존의 차량용 네트워크는 인포테인먼트 데이터를 처리하기에 대역폭이 부족하다. 그래서 등장한 네트워크가 MOST이다[1].

MOST는 멀티미디어 데이터를 전달할 목적으로 개발되어 기존 차량용 네트워크에 비하여 높은 대역폭을 제공한다. MOST는 기존의 계측 제어 장치들 사이의 통신이 아닌 다양한 멀티미디어 데이터를 전달하고 처리하기 위하여 기존의 네트워크에 비하여 복잡한 프로토콜 구조를 가지게 되었다.

MOST 프로토콜은 크게 세 개의 데이터 영역으로 구분을 할 수 있다. 컨트롤 데이터, 스트림 데이터, 패킷 데이터가 그 세 가지이다. 이 중 컨트롤 데이터를 위한 통신 프로토콜 스택을 Network Service가 정의한다. 이렇게 중요한 Network Service를 제공하는 프로그램은 MOST 메인컨트롤

* 교신저자(Corresponding Author)
논문접수 : 2009. 12. 31., 채택확정 : 2010. 02. 11
이무열, 정성문, 진현욱 : 건국대학교 컴퓨터공학부
※ 본 연구는 지식경제부의 산업원천기술개발사업과 대학 IT연구센터(ITRC) 지원 사업(#NIPA-2009-C1090-0902-0026), 그리고 교육과학기술부의 세계수준의연구중심대학 육성사업(#R33-2008-000-10068-0)의 연구결과로 수행되었음.

칩을 개발한 SMSC사에서 개발하여 상용으로 판매되는 NetService 하나뿐이다[2]. 최근 저자들에 의해서 발표된 연구들은 MOST Network Service를 포함하고 있으나 그 설계 및 구현에 대해서는 구체적으로 다루고 있지 않았다 [5-7]. 하지만 본 논문은 Network Service의 컨트롤 데이터 통신을 위한 프로토콜 스택을 지원하는 설계 및 구현을 포함하고 있으며 성능 측정을 수행한다.

차량용 인포테인먼트 시스템은 OSEK과 같은 차량용 전자장치 소프트웨어를 위한 운영 체제보다 다양한 멀티미디어 응용과 디바이스 드라이버를 제공하는 범용 운영체제(예 Linux, Windows)를 플랫폼으로 사용할 때 기존에 개발되고 유용성이 검증된 많은 소프트웨어들을 좀 더 원활하게 사용할 수 있을 것이다. 이에 본 논문은 여러 운영체제에서 구동되며, 다양한 플랫폼을 지원할 수 있도록 POSIX 인터페이스를 사용하여 이식성이 높은 Network Service 설계를 제안한다.

본 논문은 다음과 같이 구성되어 있다. 본 서론에 이어서 2장에서는 MOST에 관하여 좀 더 자세히 살펴본다. 그리고 MOST를 구성하는 중요 단위인 FBlock에 대해서 설명을 한다. 3장에서는 컨트롤 데이터 영역에 초점을 맞추어 POSIX 인터페이스를 사용한 Network Service를 제안 및 설계를 한다. 4장에서는 실제로 구현된 Network Service의 성능을 측정한다. 마지막으로 5장에서는 본 논문의 결론과 향후 연구계획을 제시한다.

II. 배경지식

MOST는 차량용 엔터테인먼트 시스템을 지원하기 위해 개발된 링 방식의 네트워크이다. 현재 MOST규격은 MOST 25, 50, 150 Mbit/sec 의 대역폭을 지원한다. 본 논문에서는 25 Mbit/sec의 규격을 가정하나 그 이후의 규격에도 큰 수정 없이 연구내용은 적용될 수 있다.

본 장에서는 MOST 네트워크에서 사용되는 스트림 데이터, 패킷 데이터, 컨트롤 데이터의 특징을 각각 살펴본다.

1. MOST 스트림 데이터

MOST의 메시지 구조는 그림1과 같은 구조를 가진다. 스트림 데이터는 라디오 방송이나 비디오 방송같은 실시간 멀티미디어 데이터를 전달 할 때 사용이 된다. 스트림데이터는 동기화된 데이터만 전달할 수 있으며 데이터가 유실될 수 있다. 또한 스트림 데이터 영역은 Boundary에 의하여 패킷 데이터 영역과 메시지 영역을 공유하는 구조이다.

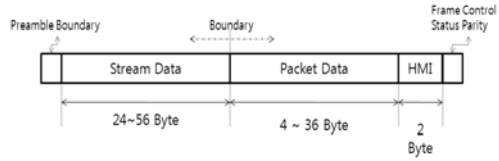

그림 1. MOST 메시지 구조
Fig. 1. MOST frame

그림 2. 컨트롤 데이터 구조
Fig. 2. Control frame structure

2. MOST 패킷 데이터

패킷 데이터는 비동기 데이터를 전달하기 위하여 사용되는 데이터 영역이다. 데이터의 손실을 파악할 수 있으며 검증을 위하여 CRC를 사용한다. 패킷 데이터는 TCP/IP같은 비동기 네트워크와 통신을 할 때 사용할 수 있다. 또한 앞서 밝힌 바와 같이 패킷 데이터 영역은 Boundary에 의하여 스트림 데이터와 메시지 공간을 함께 사용한다.

3. MOST 컨트롤 데이터

MOST 서비스는 FBlock이라고 불리는 중요 단위로 나눌 수 있다. FBlock은 특정 서비스를 제공기 위해서 사용자의 요청에 반응하고 응답하는 최소 단위이다. 또한 FBlock은 여러 영역의 데이터를 수신, 처리할 수 있으며 필요에 따라 여러 FBlock들과 통신을 한다. 또한 이러한 FBlock들이 모여서 하나의 서비스를 이루는 것을 노드라고 하고 이 노드들이 모여서 링 방식의 MOST네트워크를 형성한다.

컨트롤 데이터는 MOST FBlock들을 제어하고 상태를 모니터링하기 위하여 사용되는 영역이다. 그림1에서 마지막 HMI(human-machine interface) 영역이 16개가 모여 하나의 컨트롤 메시지를 이루고 컨트롤 데이터는 이 컨트롤 메시지 단위로 데이터가 처리된다.

컨트롤 메시지는 32바이트의 크기를 가진다. 이 중 타이밍 중재를 위한 4바이트, 발신지 주소 2바이트, 송신지 주소 2바이트, 메시지 종류 1바이트, CRC 2바이트, 전송 상태를 나타내는 4바이트를 제외한 17바이트를 사용자가 사용할 수 있으며 17바이트의 구조는 그림2와 같다. FBlockID부터 Tel Len까지의 5바이트의 헤더를 사용하여 원하는 FBlock 서비스 함수에게 데이터를 전달할 수 있다. 이 때 전달 가능한 데이터의 크기는 Tel ID의 분류에 따라 이론상 무한대의 데이터를 전달 할 수 있다[3].

3장에서 제안되는 Network Service는 컨트롤 데이터 영역을 사용하여 FBlock간의 통신을 가능하도록 해준다.

Ⅲ. 이식성 높은 Network Service 설계

본 장에서는 MOST컨트롤 데이터 통신을 위해서 POSIX 인터페이스를 사용하여 플랫폼에 제약적이지 않은 Network Service의 설계를 제안한다.

1. 전체구성

본 논문에서 제시하는 MOST Network Service는 그림 3과 같이 크게 4개의 계층으로 나눌 수 있다. 이 중 본 논문에서 구현할 주요 Network Service에 해당하는 부분은 FBlock 프레임워크 계층과 Message Core 계층이다. 본 논문에서는 Network Service 부분을 유저 영역에서 구현하도록 설계를 하였으며, 각각의 계층들 간의 IPC는 POSIX에서 정의되어있는 IOCTL과 Message Queue를 사용하였다. Network Service 각각의 계층에서 하는 역할은 다음과 같다.

Network Service의 가장 상위에는 Service Function 계층이 존재한다. 이는 개발자가 정의하는 부분으로 서비스 함수를 구현하는 부분이다. 그 하위 계층으로 FBlock 프레임워크가 존재하며 이는 FBlock을 쉽게 개발하고 생성된 FBlock들을 쉽게 관리 할 수 있도록 지원해 준다. 바로 하위 계층인 메시지 코어 계층은 MOST 네트워크로부터 수신된 각종 데이터들을 사용자가 쉽게 사용할 수 있도록 가공하여 주는 역할을 한다. 아울러 송신을 할 때에도 MOST 네트워크를 위한 MOST 프로토콜 헤더를 작성한다. 마지막으로 가장 하위 계층인 커널 영역에는 MOST 디바이스 드라이버가 존재한다. 디바이스 드라이버의 역할은 MOST 장치와 통신을 하는 것이 주된 역할이다.

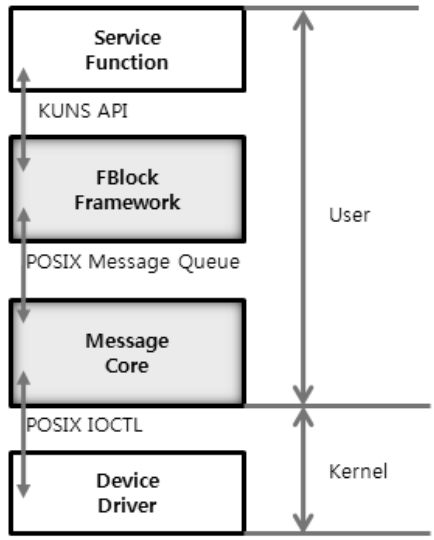

그림 3. 제안된 Network Service 전체구성도
Fig. 3. Overview of proposed network service architecture

표 1. FBlock 프레임워크 API
Table 1. APIs provided by FBlock framework

APIs	설명
KUNS_FBlock_Register	FBlock을 등록
KUNS_FBlock_Unregister	FBlock을 해지
KUNS_FunctionID_Register	특정 FBlock에 Function을 등록
KUNS_FunctionID_Unregister	특정 FBlock에 Function을 해지
KUNS_FBlock_RUN	FBlock을 실행
KUNS_Ctrl_Send	메시지 코어로 데이터 송신

2. FBlock 프레임워크 계층

FBlock 프레임워크 계층은 개발자가 FBlock을 쉽게 구현할 수 있도록 도움을 주는 역할을 한다. 표1에서 이러한 기능을 지원하는 API들을 볼 수 있

다.

하나의 FBlock은 여러 개의 서비스 함수들을 가질 수 있으며 이 함수들은 각각의 ID를 가지고 있다. 이 ID들은 컨트롤 메시지 구조에도 나타나 있는데 바로 Inst ID, Fkt ID, Optype이다(그림2).

FBlock 프레임워크는 이 헤더를 근거로 해당 서비스 함수를 찾아내고, 함수포인터를 사용하여 호출하여준다. 아울러 수신된 데이터를 해당 함수로 전달해주는 역할을 한다. 즉 FBlock 프레임워크는 수신된 데이터를 헤더를 사용하여 인자 값으로서 해당 함수로 역다중화를 수행한다.

또한 서비스 함수가 다른 MOST 장치로 데이터를 전달해야할 경우 전달할 목적지와 사용자 데이터를 프레임워크 계층으로 전달하면 이는 POSIX 큐를 사용하여 메시지코어로 전달된다.

FBlock 프레임워크는 사용자 수준의 프로세스로서 각각의 FBlock 마다 하나씩 생성된다.

3. 메시지 코어 계층

이 계층에서 하는 주된 역할은 디바이스 드라이버 계층에서부터 수신된 메시지를 FBlock 프레임워크 계층으로 넘겨주는 기능을 수행한다. 또한 FBlock 프레임워크 계층에서 수신된 사용자 데이터를 디바이스 드라이버로 전달해주는 역할을 한다. 아울러 큰 컨트롤 데이터를 지원하기 위해서 단편화 작업을 수행한다.

이론상으로 컨트롤 메시지를 통하여 전달 될 수 있는 데이터의 크기는 무한대이다. 이는 Telegram ID(Tel ID), Telegram Length(Tel Len) 항목을 통하여 무한대의 크기를 가진 데이터를 전송할 수 있다. 먼저 Tel ID가 0일 때 추가 패킷이 없음을 의미한다. 즉 하나의 패킷으로 모든 데이터가 전달되었음을 뜻한다. 하나의 패킷으로 모든 데이터를 전달 할 수 없을 때 Tel ID와 Tel Len와 데이터 카운터를 사용하여 데이터를 여러개의 패킷으로 나누어 전송한다. 먼저 Tel ID가 1일 경우 첫 번째 데이터 패킷을 의미한다. 다음으로 Tel ID가 3일 경우는 마지막 패킷을 의미한다. 그 외 Tel ID가 1, 3사이의 모든 패킷은 Tel ID값으로 2를 가진다. 이러한 방법으로 사용자 데이터는 크기 제약 없는 데이터를 보낼 수 있게 된다. 그리고 여러 개의 패킷으로 데이터를 나눌 경우 사용자 데이터 영역의 1바이트를 사용하여 현재 패킷이 몇 번째 패킷인지 순번을 저장한다. 따라서 하나의 패킷으로 모든 데이터를 전달 할 경우 사용자는 최대 12바이트를 사용할 수 있으나, 두 개 이상의 패킷을 사용할 때 각각의 패킷에서 사용자는 11바이트만 사용할 수 있다.

이 계층에서는 디바이스 드라이버로부터 수신된 패킷이 완전한 메시지를 구성하기 위해서 추가 패킷을 요구하는지 판단하여 추가 패킷이 있을 경우 기존의 패킷을 다중 리스트에 보관한다. 모든 패킷이 수신되었을 경우 패킷은 하나의 메시지로 변환되어 POSIX Message Queue를 통하여 FBlock 프레임워크 계층으로 전달된다.

FBlock이 데이터를 송신할 때, 전송하려는 데이터의 크기가 하나의 패킷에 담을 수 없을 크기일 경우 Tel ID, Tel Len을 사용하여 데이터를 적절한 크기로 나누는 역할과 각각의 패킷에 MOST 프로토콜 헤더를 붙여주는 기능을 수행한다. 이렇게 만들어진 패킷은 IOCTL을 사용하여 디바이스 드라이버로 전달된다.

본 논문에서 제안된 설계에서 메시지 코어는 사용자 수준의 프로세스로서 설계 하였으며 하나의 MOST 노드에는 하나의 메시지코어만 존재한다.

4. 디바이스 드라이버 계층

디바이스 드라이버 계층은 물리적인 MOST 네트워크로 데이터를 송수신 하는 역할을 한다. 본 논문에서 설계 및 구현을 하는데 사용된 리눅스용 MOST 디바이스 드라이버[4]에서 컨트롤 데이터는 IOCTL 시스템 호출을 통해서 데이터를 송수신하며, 스트림 데이터는 Read, Write 시스템 호출을 통해서 송수신하도록 되어 있다. 이들 시스템 호출은 모두 POSIX 인터페이스에 정의되어 있다.

하지만 기존 작성된 MOST 디바이스 드라이버는 데이터의 수신 확인 후 데이터가 없을 경우는 바로 반환되는 Nonblocking형식으로 구현되어 있었다. 따라서 응용 프로그램에서 데이터의 수신 확인을 위해서는 데이터가 수신되었는지 확인을 위한 폴링 형태로 구현되어야 하는 단점을 갖고 있었다. 본 논문에서는 이를 해결하기 위해서 기존의 디바이스 드라이버를 수정하여 수신된 데이터가 없을 경우에는 응용 프로그램이 블로킹되도록 구현하고 신규 데이터가 수신될 때 깨어나서 처리할 수 있도록 하였다. 이러한 수정은 IOCTL 인터페이스에는 영향을 미치지 않으므로 본 논문의 목표인 이식성은 유지할 수 있다.

IV. 실험 및 분석

본 장에서는 앞서 설계된 Network Service를 구현하여 이의 성능 측정을 수행했다. 구현 시 컨트롤 데이터를 전송을 할 때 너무 빠른 속도로 데이터를 전송을 할 경우 MOST 디바이스가 모든 데이터를 전송하지 못하고 데이터를 잃어버리는 현상이 측정되었다. 그래서 본 실험에서 구현된 Network Service에서는 패킷을 연속으로 보낼 때 임의의 지연시간(5ms)을 넣어서 측정하였다. 이는 본 실험환경으로 사용된 MOST OS8104칩의 명세서에도 정상적 송수신을 위해 요구되는 사항으로 명시되어 있다.

1. 실험 환경 및 목표

실험에 사용된 시스템의 특성은 표2와 같다. 실험은 본 논문에서 설계한 Network Service를 사용하여 각각의 시스템에 FBlock들을 만들고 하나의 FBlock에서 데이터를 생성하여 다른 FBlock으로 데이터를 전송 한 뒤 이를 다시 데이터를 생성한 FBlock으로 재전송하여 데이터가 도달하기까지 걸리는 시간을 측정하였다. 이 실험을 통하여 본 논문에서 제안하고 설계 및 구현한 Network Service가 자체 오버 헤드가 크지 않음을 보이고 올바르게 구동함을 보인다.

표 2. 실험용 시스템
Table 2. Experimental systems

	MOST 노드1	MOST 노드2
Type	산업용 임베디드 보드	PC
CPU	Pentium M	Intel Quad Q8200
OS	SUSE 10.3	Fedora 9
MOST	MOST PCI Tool Kit 25o NIC	

2. 실험 결과 분석

하나의 패킷에서 사용가능한 최대 사용자 데이터 크기는 12Byte이다. 그림 4에서 알 수 있듯이 하나의 패킷이 왕복하는 데 걸리는 시간은 약 4ms 정도이다. N(>=2)개의 패킷에서 사용가능한 최대 사용자 데이터의 크기는 식(1)과 같다.

$$N*11_{Bytes} \qquad (1)$$

실험에서 사용된 MOST 디바이스의 경우 너무 빠른 속도로 데이터를 전송하려고 할 경우 데이터를 전송하지 못하고 잃어버리는 현상이 측정이 되었다. 그래서 구현 시 사용자 데이터가 커서 두 개 이상의 패킷으로 전달해야 할 경우 마지막 패킷을 제외한 모든 패킷에 전송을 위한 5ms의 지연 시간을 두었다.

먼저 N개의 패킷을 전송하는데 걸리는 예상 시간을 알아보자. N개의 패킷을 전송을 할 때 마지막 패킷은 지연 시간이 없다. 그 외 모든 N-1개의 패킷은 5ms의 지연 시간을 가진다. 패킷은 왕복하기 때문에 양쪽 시스템의 지연 시간을 가진다. 그러므로 지연 시간은 아래 식(2)와 같이 2배가 된다.

$$4+((N-1)*5)*2 \qquad (2)$$

그림 4와 위의 수식을 비교하면 실험 결과와 수식이 거의 일치함을 알 수 있다. 이는 본 논문에서 설계하고 구현한 Network Service가 오버헤드가 거의 없이 올바르게 동작함을 보인다.

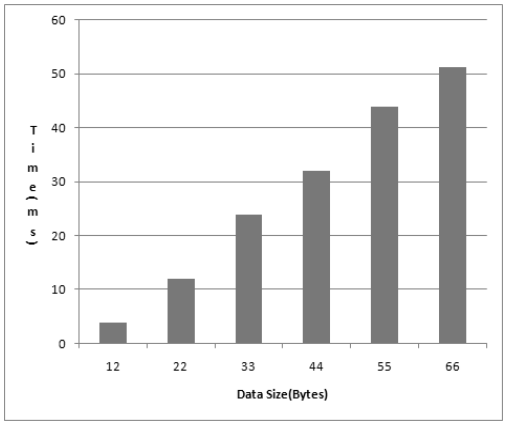

그림 4. 메시지 크기별 왕복 통신시간
Fig. 4. Round-trip time

V. 결론 및 향후 연구 계획

본 논문은 차량용 인포테인먼트 시스템을 위하여 제안된 MOST Network Service의 설계 및 구현을 제안하였다. 특히 POSIX 인터페이스를 사용하여 여러 플랫폼에 쉽게 이식 할 수 있도록 하였으며, MOST 개발자가 FBlock을 쉽게 생성, 관리할 수 있도록 각종 편리한 API들을 제공한다. 또한 다

양한 크기의 컨트롤 메시지를 지원할 수 있게 하였다. 그리고 본 논문에서 제안되고 구현된 Network Service의 성능을 측정하여 자체 오버헤드가 크지 않음을 보였다.

향후 연구 계획은 구현된 Network Service를 리눅스가 아닌 타 플랫폼(Windows)에서 구동시킬 예정이다. 또한 이식성이 아닌 성능에 중점을 맞춘 커널 수준의 Network Service를 설계하고 구현할 계획이다. 그리고 본 논문에서는 컨트롤 데이터 영역만 지원하지만, 앞으로 스트림 데이터와 패킷 데이터 또한 지원을 할 수 있도록 확장할 계획이다.

참고문헌

[1] MOST Cooperation, "MOST specification", Rev 3.0, May. 2008.

[2] http://www.smsc.com

[3] Andreas Grzemba, "MOST - The automotive multimedia network", Franzis 2008.

[4] B. Walle, "Development of a linux driver for a MOST interface and porting to RTAI", Diploma Thesis, Sep. 2006.

[5] Mu-Youl Lee, Sung-Moon Chung, and Hyun-Wook Jin, "Automotive network gateway to control electronic units through MOST network", IEEE ICCE, Jan. 2010.

[6] 정성문, 이무열, 진현욱, "실시간 통신 도메인을 고려한 차량용 네트워크 게이트웨이", 한국정보처리학회 추계학술발표대회 논문집, 2009.

[7] 이무열, 정성문, 진현욱, "리눅스기반 차량용 MOST-CAN 네트워크 게이트웨이 설계 및 구현", 한국컴퓨터종합학술대회(KCC 2009)논문집, 2009.

저 자 소 개

이 무 열

2008년 8월 : 건국대학교 컴퓨터 공학과 학사.
2008년 9월~현재, 건국대학교 대학원 정보통신공학과 컴퓨터공학 석사과정.
관심분야 : 임베디드소프트웨어, 실시간운영체제.
Email : zlemy@konkuk.ac.kr

정 성 문

2007년 : 건국대학교 산업공학과 학사.
2008년 : (주)미라콤 테크놀로지 연구원.
2009년~현재, 건국대학교 정보통신공학과 컴퓨터공학 석사과정.
관심분야 : 임베디드소프트웨어, 실시간운영체제.
Email : smchung@konkuk.ac.kr

진 현 욱

1997년 2월 : 고려대학교 전산학 학사.
1999년 2월 : 고려대학교 전산학 석사.
2003년 8월 : 고려대학교 통신시스템공학 박사.
2003년 9월~2006년 1월 : 미국 오하이오 주립대학교 연구원.
2006년 3월~2010년 2월 : 건국대학교 컴퓨터공학부 조교수.
2010년 3월~현재, 건국대학교 컴퓨터 공학부 부교수.
관심분야 : 운영체제, 차량 및 항공기용 소프트웨어 플랫폼.
Email : jinh@konkuk.ac.kr