공유메모리를 통한 신뢰성 있는 데이터 교환 프로토콜

김 영 신, 권 욱 현
서울대학교 전기공학부 제어정보시스템연구실
kys@cisl.snu.ac.kr, whkwon@cisl.snu.ac.kr

Reliable Data Exchange Protocol Using Shared Memory

Young Shin Kim, Wook Hyun Kwon
Control Information Systems Lab., School of Electrical Engineering, Seoul National University, Korea
kys@cisl.snu.ac.kr, whkwon@cisl.snu.ac.kr

Abstract

This paper suggests two data exchange protocols (DEPs) using shared memory. They support reliable data exchanges among components which are located in the same rack and use the same backplane bus and shared memory.

1 서론

분산 제어 시스템은 공정의 분산 및 최적 제어를 통한 생산성 향상은 물론, 에너지 절감, 생산 원가 절감, 인력 절감 등의 제반 효과를 얻을 수 있을 뿐만 아니라 기존 중앙 집중적 제어 형태의 제반 단점인 전 가동 라인의 중단이라는 문제를 해결할 수 있어 공장 자동화의 확산과 함께 국내외 전 산업계에 보편적으로 채용되었다. 분산 제어 시스템은 프로세스의 대규모화에 따른 제어 및 운전, 감시의 복잡화를 효율적으로 처리하기 위해 제어의 분산을 통한 최적화, 정보의 집중화를 기본 개념으로 한다. 따라서 분산 시스템은 여러 노드들이 통신망을 통해 연결된다. 원격 I/O로부터 입력을 받아서 연산을 수행한 후 원격 I/O로 출력을 내는 노드들이 있고, 전체 시스템 구성 요소의 프로세스 정보를 총괄적으로 관리하는 노드, 프로세스의 상황을 그래픽 화면으로 보여주는 MMI (Man Machine Interface)를 담당하는 노드, 프로세스 정보를 다양한 형태로 관리하는 데이터베이스 노드, 그리고 시스템 관리자가 시스템을 수정, 변경 등을 할 수 있도록 제공하는 노드 등이 있다.

각 노드는 보통 시스템 제어 모듈, 통신 모듈, 연산 모듈, 메모리 모듈 등 여러 구성 요소로 이루어진다. 각 구성 요소들은 하나의 렉(rack)에서 하나 이상의 슬롯(slot)을 차지한다. 각 구성 요소들은 렉에서 제공하는 백플래인 버스와 공유 메모리 등을 사용하여 데이터를 공유하거나, 데이터를 교환한다. 산업계에서 사용하는 백플래인 버스로는 VMEbus, S-bus 등 다양하다. 이 중에서 VMEbus는 산업계에서 가장 널리 쓰이는 시스템 버스 중의 하나이다. VMEbus에 대한 연구로는 통신망의 응용[1, 2], 데이터 수집 및 분석[3, 4, 5], 영상 수집 및 분석[6], 계측 및 시스템의 제어[7, 8] 등 대부분의 분산 제어 시스템이 해당된다. 대부분의 연구가 VMEbus의 응용에 있으며 VMEbus를 보다 신뢰성 있고 효율적으로 사용하려는 연구는 거의 없었다. 그리고 공유 메모리를 통한 데이터의 공유에 대한 연구는 많이 이루어졌으나, 공유 메모리를 통한 구성 요소들 사이의 신뢰성 있는 데이터 교환에 대한 연구는 없었다. 본 논문에서는 VMEbus를 사용함에 있어 보다 신뢰성 있는 데이터 교환 프로토콜을 제안하고자 한다.

2 대상 시스템

본 논문에서 대상으로 하는 시스템은 VMEbus를 사용하고, VMEbus의 마스터인 HOST와 네트워크 접속장치(NIU), 연산 보드, 등 여러 슬레이브들로 구성된다. 각 구성 요소들 사이에는 공유 메모리를 이용하여 데이터를 공유하거나, 교환한다. 본 논문에서는 특히 분산 제어 시스템을 대상으로 하여 HOST와 NIU 사이의 데이터 교환을 대상으로 한다. 하지만 본 논문의 결과는 대부분의 공통의 백플래인을 사용하는 구성 요소들 사이의 데이터 교환에 사용될 수 있다.

데이터베이스로 사용하는 공유 메모리의 경우에는 보통 HOST와 NIU가 모두 공유 메모리에 쓰고 공유 메모리로부터 읽는다. 이 경우에는 상호 배제(mutual exclusion)를 위해 공유 메모리 관리에 세마포어 등이 필요하다. 하지만 통신을 위한 공유 메모리에서는 한 쪽에서 데이터를 쓰고 다른 쪽에서 읽어 가는 형태로 공유 메모리가 사용된다. 이 경우에는 보통 공유 메모리를 분리해서 관리한다. 한 쪽에서 쓰고 다른 쪽에서는 읽어 가는 방식이다. 공유 메모리는 HOST에서 NIU로 보내는 영역과 NIU에서 HOST로 보내는 영역으로 나누어져 있다. HOST에서 NIU로 보내는 영역은 HOST에 의해 관리되며, HOST만이 쓸 수 있고 NIU는 읽을 수만 있다. 마찬가지로 NIU에서 HOST로 보내는 영역은 NIU에 의해 관리되며, NIU만이 쓸 수 있고 HOST는 읽을 수만 있다. 이렇게 함으로써 공유 메모리에 있는 데이터를 쉽게 관리할 수 있으며, 신뢰성을 확보할 수 있다.

또한 공유 메모리를 사용한 데이터의 교환은 기본적으로 확인 메커니즘(acknowledgement mechanism)을 사용한다. 일정 시간이 지나도록 확인이 오지 않으면 재시도를 한다. 재전송을 하는 경우에는 공유 메모리에 데이터를 다시 쓰지는 않고 새로운 데이터가 공유 메모리에 있음을 알린다. 공유 메모리에 새로운 데이터가 있음을 알리는 것과 확인을 하는 것은 모두 인터럽트를 사용한다고 가정한다.

3 문제 설정

공유 메모리를 통하여 데이터를 교환할 때 다음과 같은 사항들이 고려되어야 한다.

확인 메커니즘 공유 메모리를 통해 HOST와 NIU가 데이터를 교환할 때 논리적인 오류가 없어야 한다. 또한 데이터의 교환이 유한 시간(데드라인) 안에 이루어져야 하며, 이루어지지 못했을 경우 이 사실을 확인할 수 있어야 한다.

상호 배제(Mutual Exclusion) 데이터의 완결성을 보장하기 위해서는 한 쪽에서 공유 메모리에 쓸 때, 동시에 다른 쪽에서 읽어가지 않도록 한다. 물론 한 쪽에서 읽어갈 때 동시에 다른 쪽에서 써서도 안된다. 양쪽에서 동시에 읽어 가는 것은 데이터의 완결성에 영향이 없다.

독립성(Independability) 공유 메모리를 통해 데이터를 교환하는 과정에서 한 쪽의 지연과 문제가 다른 쪽의 동작에 영향을 주어서는 안된다. 이 문제는 데이터를 교환하는 두 개 이상의 구성 요소에서 흔히 발생한다. 특히 단일 태스크 운영체제를 사용하는 경우에 많이 발생하는 문제로 데드 락(Dead lock)이 발생할 수도 있다.

본 논문에서 고려하는 대상 시스템은 현재 산업계에서 사용되고 있다. 공유 메모리를 분리해서 사용하고, 인터럽트를 이용하여 새로운 데이터의 존재를 알리고, 또한 데이터를 읽어 갔음을 확인하는 과정으로 이루어진다. 그리고 일정 시간 이상 데이터 수신에 대한 확인이 없는 경우 재전송을 시도한다. 언뜻 보면 문제가 없어 보인다. 하지만 다음과 같은 두 가지 문제가 발생할 수가 있다.

우선 새로운 데이터가 왔음을 알리는 과정에서 데이터에 순서가 없어 발생할 수 있는 문제이다. 본 논문에서는 이를 번호 없는 데이터의 문제라고 하겠다. 그림 1을 보면, HOST에서 NIU로 data_0을 보냈을 경우, 타임아웃이 되도록 확인이 오지 않으면 HOST는 data_0을 다시 보낸다. 그런데 NIU가 늦게 data_0에 대한 확인을 하였다. 확인을 하고 난 다음 재전송된 데이터를 받았을 때 NIU는 이것이 확인을 한 데이터가 재 전송된 것인지 새로운 데이터가 전송된 것인지를 구별할 수 없다. 이렇게 되면 타임아웃을 잡는 것이 중요한 이슈가 되고 이는 쉽지 않은 문제가 된다. 또한 데이터의 길이마다 타임아웃이 달라질 수 있기 때문에 본질적으로 문제를 해결할 수 없다.

이 문제를 해결하기 위해서 데이터에 번호를 붙여서 보낸다.

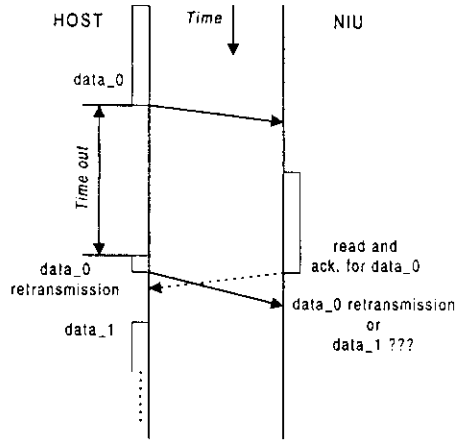

그림 1 번호 없는 데이터 문제

이 경우에 NIU에서는 HOST가 보낸 데이터가 재 전송된 데이터인지 새로운 데이터인지를 번호를 보고 알 수 있다.

다음 문제로, 이 번에는 HOST 측에서 NIU 측으로부터 받은 확인에 대해 혼란이 발생할 수 있다. 본 논문에서는 번호 없는 확인의 문제라고 한다. 그림 2를 보면 data_0을 보낸 후 타임아웃까지 확인이 오지 않아 data_0을 다시 보낸다. 이 때 NIU에서는 확인을 보내고, 재 전송된 data_0을 받는다. NIU 측에서는 data_0이 재 전송된 이유가 타임아웃 때문인지 아니면 data_0에 대한 확인이 HOST에 제대로 가지 않은 것 때문인지를 모르기 때문에 다시 data_0에 대한 확인을 시도한다. 한편 HOST 측에서는 data_0에 대한 확인을 받았으므로 data_1을 공유 메모리에 쓰고 NIU에 전송을 알린다. 그리

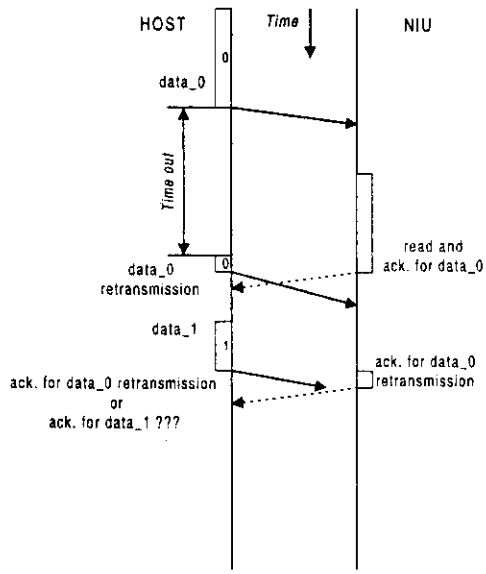

그림 2 번호 없는 확인의 문제

고 나서 NIU로부터 확인을 받는다. HOST는 이 확인이 재 전

송된 data_0에 대한 확인인 지 data_1에 대한 확인인 지를 알 수 없다. 이 때, 불행히도 data_1에 대한 전송이 NIU에 제대로 도달하지 못했다면 시스템은 예상하지 못한 방향으로 동작을 할 수 있다. 이 문제를 해결하기 위해서는 확인에도 번호를 매겨 어떤 데이터에 대한 확인임을 명시해야 한다.

4 신뢰성 있는 데이터 교환 프로토콜

본 논문에서는 3절에서 논한 문제를 해결하기 위해서 두 가지 데이터 교환 프로토콜을 제안한다. 본 논문에서 제시하는 알고리즘에서, 확인을 하기 위한 시스템의 부담을 줄이기 위해서 확인을 피기백으로 보내는 방식을 고려할 수 있으나 피기백 방식은 수신 측에서 송신 측으로 보낼 데이터가 있을 때까지 일정시간 (타임아웃) 대기를 해야하는 시간적인 손해를 감수해야 한다. 따라서 실시간 시스템의 데이터 교환에는 적당하지 않다고 생각되어 데이터를 받으면 확인을 하는 과정을 따로 가진다.

4.1 정지 후 대기 (Stop and Wait) 방법

본 논문에서는 번호 없는 데이터 문제와 번호 없는 확인 문제를 해결하기 위해서 데이터와 확인에 번호를 매겼다. 데이터에는 데이터의 순서 번호(SN)를 매겼고, 확인에는 기다리는 데이터의 번호(RN)를 매겼다. 처음 시작할 때 보내는 측과 받는 측에서 SN과 RN을 초기화한다. 본 논문에서는 알고리즘 1을 제안한다. 알고리즘 1은 데이터를 보내고 확인을 받을 때까지 추가적인 데이터를 보내지 않는다. 이 방식은 공유 메모리의 크기가 작거나, 공유 메모리를 통해서 주고받는 데이터들이 시간적인 연관성을 가지고 있는 경우에 적당하다. 이 알고리즘에서는 SN과 RN이 0 또는 1의 값을 가진다.

알고리즘 1.1 (보내는 측)

단계 1. SN을 0으로 한다.

단계 2. 보낼 데이터가 발생하기를 기다린다. 보낼 데이터가 있으면 데이터를 공유 메모리에 쓰고 SN을 데이터에 매긴다.

단계 3. 받는 측에 알린다.

단계 4. 받는 측으로부터 SN과 다른 RN을 받으면 SN을 RN으로 하고 단계 2로 간다. 만일 타임아웃 동안 SN과 다른 RN을 받지 못하면 단계 3으로 간다.

알고리즘 1.2 (받는 측)

단계 1. RN을 0으로 하고 단계 2를 반복한다.

단계 2. RN과 같은 SN을 가지는 데이터를 받으면 RN을 바꾸어 (RN이 0인 경우 1로, 1인 경우 0으로) 데이터를 보낸 측으로 확인을 보낸다.

이 방식은 확인을 받기 전에는 다른 데이터를 보내는 것을 대기하고 있어야 하기 때문에 다른 처리를 하지 못하는 경우도 발생할 수 있다. 이는 상대방의 동작에 영향을 많이 받는다고 말할 수 있다. 안전관련 시스템에서는 구성 요소들 사이에 독립성이 최대한 보장되어야 하기 때문에 이러한 상황은 바람직하지 않다. 이런 상황을 줄이기 위해 본 논문에서는 Go back n 방식을 제안한다.

4.2 N회 되돌아가기 (Go back n) 방법

공유 메모리의 크기가 충분히 크고 HOST에서 NIU로 보내는 데이터의 관련성이 적은 경우에는 확인 응답을 받기 전에 다음 데이터를 공유 메모리에 쓰고 NIU로 알리는 것이 효율적일 수 있다. 이 경우에는 HOST에서 NIU로 보내는 데이터에 대한 공유 메모리 영역이 세분되어야 하며 공유 메모리 관리가 필요하다. 공유 메모리 관리에서는 확인을 받지 않은 데이터가 쓰여있는 영역에 대한 보호와 확인을 받은 데이터가 쓰여있는 영역의 해제 그리고 새로운 데이터를 쓰기 위한 영역의 할당 등을 한다. 알고리즘에서 사용하는 m은 n보다 큰 정수이다. m을 n+1로 잡으면 쉽다.

알고리즘 2.1 (보내는 측)

단계 1. 모듈로 m 변수 SN_{min}과 SN_{max}를 0으로 한다.

단계 2. 단계 3, 4, 5를 순서 없이 반복한다.

단계 3. $(SN_{max} - SN_{min}) \bmod m < n$ 이고, 보낼 데이터가 있으면, SN_{max}를 그 데이터의 SN으로 공유 메모리에 쓴다. 그리고 SN_{max}를 $(SN_{max} + 1) \bmod m$ 으로 증가시킨다.

단계 4. RN을 받았을 때,

$(RN - SN_{min}) \bmod m \geq (SN_{max} - SN_{min}) \bmod m$

이면, SN_{min}을 RN으로 설정한다.

단계 5. $SN_{min} \neq SN_{max}$이고 타임아웃이 발생하면,

SN_{min}에 해당하는 데이터의 교환을 받는 측에 다시 알린다.

알고리즘 2.2 (받는 측)

단계 1. 모듈로 m 변수 RN을 0으로 한다.

단계 2. SN이 RN과 같은 데이터를 받을 때마다, RN을 (RN + 1) mod m 으로 증가시킨다.

단계 3. 타임아웃 안에 데이터를 보낸 측으로 RN을 포함한 확인을 보낸다.

구현 내용

VME 백플레인을 대상으로 하여 구현하였다. HOST와 NIU는 실시간 운영체제인 VRTXsa가 장착되어 멀티 태스킹 환경을 제공하며 프로세스간의 우선 순위에 기반한 태스크 스케줄링이 가능하다. 데이터 교환 프로토콜 프레임은 데이터 순서 번호(SN), 확인 번호(RN), 데이터의 종류, 데이터의 길이, 데이터, 오류 확인을 위한 체크 섬(check sum)으로 구성하였다.

공유 메모리 접근 시간을 줄이기 위해 4 바이트 단위로 읽고 쓴다. 공유 메모리를 HOST와 NIU가 같이 사용하도록 중재 회로를 구현하였다. 이 중재회로에서는 HOST와 NIU가 공유 메모리를 사용할 때 한 사이클씩 번갈아 사용할 수 있도록 설계하였다. 물론 동시에 공유 메모리를 접근하지 않는 경우에는 한 쪽이 계속 사용한다. 이렇게 함으로써 한 쪽에

서 긴 데이터를 공유 메모리에 쓸 경우에 다른 쪽에서 짧은 데이터를 공유 메모리에 쓰기 위해 대기해야 하는 시간을 줄인다. 공유 메모리에 데이터를 쓰고 받는 측에 알릴 때 인터럽트를 사용한다. 또한 데이터를 읽은 후에 보낸 측에 확인을 알릴 때도 인터럽트를 사용한다.

5 결론

많은 시스템에서 공유 메모리를 사용하여 데이터를 교환하고, 공유한다. 데이터를 공유하는 경우에 대한 연구는 많이 되어 있으나 데이터 교환에 대한 연구는 거의 되어있지 않다. 본 논문에서는 신뢰성 있는 데이터 교환 프로토콜을 제안하였다. 기존의 데이터 교환 프로토콜이 가지고 있던 번호 없는 데이터의 문제와 번호 없는 확인의 문제를 해결하였다. 백플래인 버스와 공유 메모리를 사용하여 이루어지는 구성 요소들 사이의 데이터 교환의 신뢰성을 보다 높였다. 또한 노드의 각 구성 요소들 사이의 의존관계를 줄임으로써 각 구성 요소들의 성능을 최대한 사용할 수 있도록 하였다.

본 논문에서 제시한 알고리즘은 실시간 운영체제를 사용하는 VMEbus 시스템에서 구현되어 실험하였다. 또한 제시한 알고리즘은 VMEbus 뿐만 아니라 대부분의 백플래인 버스 시스템에 적용될 수 있다.

제안한 데이터 교환 프로토콜의 정확한 성능을 보이기 위해 성능 해석 및 모의 실험 등이 앞으로 필요하다.

참고문헌

[1] Kimura-T, "VME and Network Applications for the JT-60U Control-System", *NUCLEAR INSTRUMENTS & METHODS IN PHYSICS RESEARCH SECTION A-ACCELERATORS SPECTROMETERS DETECTORS AND ASSOCIATED EQUIPMENT*, Vol 352, Iss 1-2, pp 125-127, 1994

[2] Cancelo-GIE Catalfo-JM Mayosky-MA, "A VME Based Map Node Running Under Os9 Operating System", *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, Vol 36, Iss 5, pp 1612-1615, 1989

[3] Zwoll-K Drochner-M Erven-W Holzer-J Kopp-H Loevenich-HW Wustner-P Watzlawik-KH Brummund-N Karnadi-M Nellen-R Stock-J Dienel-S Leege-KH Oehme-W, "Flexible Data-Acquisition System for Experiments at COSY", *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, Vol 41, Iss 1, pp 37-44, 1994

[4] Borome-N Bossu-Y Douet-R Harroch-H Trankhanh-T, "A VME Multiprocessor Data Acquisition-System Combining a UNIX Workstation and Real-Time Microprocessors", *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, Vol 37, Iss 4, pp 1514-1519, 1990

[5] Bassini-R Boiano-C Brambilla-S Cevoli-R Malatesta-M, "A VMEbus Based Data-Acquisition System for a Multielement Detector Array", *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, Vol 43, Iss 3, pp 1778-1783, 1996

[6] Epaud-F Verdier-P, "VME Image Acquisition and Processing Using Standard TV CCD Cameras", *NUCLEAR INSTRUMENTS & METHODS IN PHYSICS RESEARCH SECTION A-ACCELERATORS SPECTROMETERS DETECTORS AND ASSOCIATED EQUIPMENT*, Vol 352, Iss 1-2, pp 226-231, 1994

[7] Lim-D Seraji-H, "Configuration Control of a Mobile Dexterous Robot - Real-Time Implementation and Experimentation", *INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, Vol 16, Iss 5, pp 601-618, 1997

[8] Pfeifer-T Park-HS Thrum-H, "Flexible Integration of Various Fieldbus and Sensor/Actuator Bus Systems into Machine-Tool Control", *MICROSYSTEM TECHNOLOGIES*, Vol 3, Iss 4, pp 191-198, 1997