고속병렬컴퓨터의 크로스바 네트웍 라우터 에뮬레이터 설계 및 구현

정 성 인[†] · 이 재 경[†] · 김 해 진[††] · 임 기 욱[†]

요 약

본 논문은 SPAX 시스템의 Xcent-Net[2] 크로스바 네트웍 라우터인 XNIF[3] 하드웨어 동작에 대한 에뮬레이터의 설계 및 구현을 제시한다. 에뮬레이터 설계는 XNIF 보드의 사양과 기능을 기반으로 하고, 특히 XNIF 보드의 메시지 처리에 대한 병렬성, MISIX[4] IPC간 인터페이스와 에뮬레이터 환경에서의 메시지 전송 방법에 중점을 두고 이루어졌다. 이러한 설계는 MISIX 마이크로 커널 운용체제에 구현하였다. 그리고 에뮬레이터의 동작 검증 및 MISIX IPC 기능 시험은 Ethernet으로 연결된 두 대의 Compaq Proliant 4000 시험 시스템에서 이루어졌다. MISIX 개발과 SPAX 하드웨어 개발이 병행 진행되는 상황에서, XNIF 에뮬레이터 개발로 시험 시스템에서 Xcent-Net에 대한 MISIX IPC의 설계 및 구현이 가능했다.

The Design and Implemenation of Crossbar Network Router Emulator for SPAX[1]

Sung-In Jung[†] · Jae-Kyoung Lee[†] · Hae-Jin Kim[††] · Ki-Wook Rim[†]

ABSTRACT

This paper presents the design and implementation of emulator which emulates Xcent-Net crossbar network router board called XNIF in SPAX system. This design is based on the specifications of XNIF board, and it mainly focuses on concurrent message handling, message passing interfaces between emulator and MISIX IPC, and message transmission mechanism. Based on this design, we implemented an emulator within MISIX microkernel operating systems. Besides, the correctness of emulator and MISIX IPC is verified on two Compaq Proliant 4000 systems connected via Ethernet. Finally, this emulator development enabled us to design and implement the MISIX IPC for Xcent-Net in test-bed system, in parallel with XNIF hardware development.

[1] SPAX(Scalable Parallel Architecture computer based on X-bar network): 정보통신부 및 과학기술처의 지원으로 한국전자통신연구원이 주관하여 연구 개발하고 있는 고속병렬컴퓨터(주전산기 IV)의 시스템 명칭이다.

[2] Xcent-Net(10x10 Crossbar Network): SPAX 시스템의 상호 연결망 명칭으로 한국전자통신연구원에서 독자 설계한 계층적인 크로스바 연결망이며, X는 Crossbar를 의미하고 cent는 Xcent-Net이 10x10 포트를 가지므로 100를 의미한다.

[3] XNIF(Xcent-Net InterFace): Xcent-Net의 라우터(router) 보드 명칭이다.

[4] MISIX(Microkernel-based Single system Image uniX): 한국전자통신연구원이 개발중인 SPAX용 마이크로 커널 기반의 운용체제 명칭이다.

† 정 회 원: 한국전자통신연구원 시스템 S/W 연구실
†† 종신회원: 한국전자통신연구원 시스템 S/W 연구실
논문접수: 1996년 5월 25일, 심사완료: 1997년 11월 18일

1. 서 론

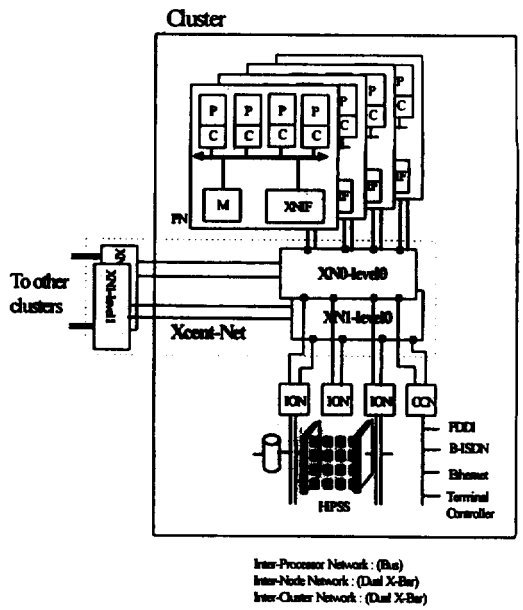
과거 10년의 하드웨어 발전은 향후의 빠른 하드웨어 발전을 예상할 수 있다. 그리고 사용자는 현재의 컴퓨팅 환경에서 해결할 수 있는 문제보다 더 새롭고 복잡한 문제에 직면하고 있으며, 시스템 설계자들도 이러한 문제를 해결할 수 있는 시스템 개발에 관심을 두고 있다. 이러한 응용 분야의 프로그램과 데이터의 크기는 점점 증대되어, 신뢰성 있는 고속 시스템을 요구한다[1]. 고성능 마이크로프로세서 출현은 다중처리기 구조에서 고속 연산 병렬처리를 가능하게 하고, 고속의 통신 기술로 클라이언트/서버 개념에 근거한 다운사이징을 출현 시킴으로써 기존의 전통적인 중앙 집중식을 분산 처리화하고 있다. 즉, 기존의 단일처리기 및 다중처리기 환경에서 벗어나 MPP (Massively Parallel Processor)와 클러스터 기반의 멀티컴퓨터 환경으로 확장되어 가고 있다[2].

고성능 멀티컴퓨터의 출현으로 운영체제 설계자들은 시스템의 확장성, 가용성, 융통성 등의 면에서 기존의 통합형(Intergrated Kernel) 기반의 운용체제 구조를 재고려하게 되었다. 최근의 연구들로[3, 4, 5, 6] 마이크로 커널 기반의 운영체제 구조가 적합하다고 밝혀졌다. 현재, Chorus systems사의 Chorus/MiX와 OSF RI의 OSF/1 AD 2.0은 마이크로 커널 기반의 운영체제 분류에 속한다. Chorus/MiX[7]는 Chorus 마이크로 커널에 System V 4.0 기능을 서버화한 운용체제이고, OSF/1 AD 2.0[8]은 CMU의 MACH[9] 마이크로 커널위에 OSF/1 1.3의 기능을 제공하는 운영체제로 OPUS(Open Parallel Unisys Server), Intel Paragon과 같은 MPP 시스템에 각각 수행되고 있다.

클러스터 기반의 멀티컴퓨터 시스템 및 마이크로 커널 기반의 운영체제 환경에서의 시스템 확장성은 매우 용이하지만, 여러 시스템을 연결하는 고속의 네트웍 및 효율적인 소프트웨어가 지원되어야 우수한 시스템 성능을 발휘할 수 있다. 현재 개발 중인 클러스터 기반의 멀티컴퓨터 시스템인 SPAX에서는 고속의 크로스바 네트웍으로 Xcent-Net를 지원한다. 그리고 SPAX에 탑재될 마이크로 커널 기반의 운용체제인 MISIX는 Xcent-Net를 이용하여 다양한 통신을 하게 된다. 이 SPAX와 MISIX 개발이 동시에 진행되고 있어, 이 SPAX와 MISIX 개발이 동시에 진행되고 있어, MISIX 개발자들은 시험 시스템인 Ethernet으로 연결된 Compaq Proliant 4000 시스템을 이용하고 있다. 따라서, 본 에뮬레이터 개발의 동기는 SPAX에서 지원하는 메시지 통신 방법과 유사한 환경에서 MISIX IPC 및 커널의 상위 기능들을 개발하기 위해서이다.

본 논문에서는 Xcent-Net의 라우터 보드인 XNIF를 시험 시스템에서 에뮬레이션한 환경은 MISIX IPC 및 커널의 상위 기능 구현에 큰 도움이 되었다. 사용된 시험 시스템은 UnixWare/MK가 수행되는 Compaq Proliant 4000으로 하였고, 두 대의 Compaq 시스템을 Ethernet으로 연결하여 구성하였다. 에뮬레이터 개발 검증은 시스템의 부팅 및 원격 시스템간의 통신 명령어를 통해서 이루어졌다.

성공적인 에뮬레이터 개발로, 다음과 같은 세가지의 결과를 얻었다. 첫째, 시험 시스템에서 Xcent-Net 기반의 MISIX IPC(Inter-Process Communication) 메시지 송수신 기능을 구현하였다. 이 MISIX IPC는 SPAX 시스템에서 수정 없이 재사용할 수 있다. 둘째, 메시지 송수신을 위해 제공되는 XNIF 인터페이스들을 마이크로 커널 관점에서 검증하였다. 이 과정을 통하여 XNIF 기능 및 규격을 미리 수정 및 보완할



(그림 1) SPAX 시스템 구조
(Fig. 1) SPAX System Architecure

수 있었다. 마지막으로, 커널 상위 기능들을 SPAX 시스템상의 통신 방법을 이용하여 구현하였다. 이상과 같은 이점으로, MISIX 개발 일정에 도움이 되었다.

본 논문의 구성은 다음과 같다. 2장은 SPAX 시스템 구조 및 Xcent-Net의 인터페이스인 XNIF를 소개한다. SPAX에 탑재될 운영체제인 MISIX의 특징을 기술한다. 3장은 라우터 에뮬레이터의 설계 개념 및 고려 사항을 기술하고, 4장에서는 상세한 구현 내용 및 에뮬레이터 구조에 대한 것을 기술한다. 5장에서는 에뮬레이터 검증에 대해서 언급한다. 마지막은 결론으로 맺는다.

2. 배 경

이 절에서는 우리가 개발 중인 하드웨어 및 운영체제인 개략적인 정보와 에뮬레이션 대상인 Xcent-Net 라우터 보드에 대해서 기술한다. 더 자세한 내용은 참고 문헌에 있다[10, 11, 12].

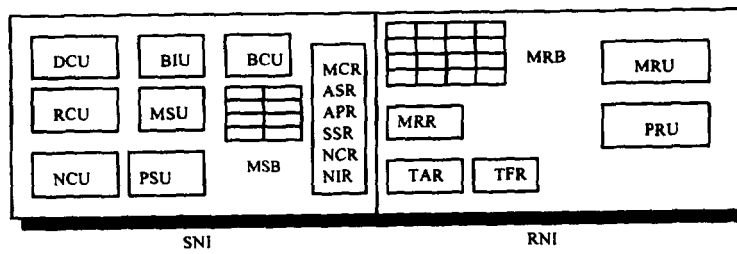
2.1 SPAX 구조

한국전자통신연구원에서는 클러스터 기반의 멀티컴퓨터인 SPAX (Scalable Parallel Architecture computer based on X-bar network)[10]와 SPAX에 탑재될 마이크로 커널 기반의 운영체제인 MISIX(MIcrokernel-based Single system Image uniX) [11]를 개발하고 있다. SPAX 시스템은 다중처리기 노드들이 고속 내부 네트웍에 연결되어 클러스터를 이루는 계층적 구조를 갖는다. (그림 1)의 SPAX 시스템은 최대 16개의 클러스터들로 구성될 수 있다. 각 클러스터에는 계산 노드(PN), 입출력 노드(ION), 통신 접속 노드(CCN)로 최대 8 노드까지 다양한 구성을 갖을 수 있다. 한 클러스터의 적합한 구성은 4개의 PN, 3개의 ION, 1개의 CCN이다. 내부 네트웍인 Xcent-Net은 10x10 크로스바 스위치로 구성되며, 각 클러스터들은 2GBPS Xcent-Net으로 이중화 연결되며, 클러스터 내의 노드들도 같은 형태로 연결된다. 각 노드는 최대 4개의 P6 마이크로프로세서, 1GB 지역 메모리, 라우팅 보드인 XNIF(Xcent-Net InterFace)[12]로 구성된다.

2.2 XNIF 구조

XNIF 보드는 각 노드에 부(daughter) 보드로 장착되어, Xcent-Net 프로토콜에 따라 다양한 형태의 메시지 송수신을 담당한다[12]. 이 보드느 송신부인 SNI (Send Network Interface)로 구성된다. 시스템의 신뢰성과 메시지 전송율을 높이기 위해 이중화된 Xcent-Net에 대해 송수신 각각 두개의 포트를 갖는다. 전송 방식을 점대점 전송, 멀티캐스트 전송, 브로트캐스트 전송 기능을 제공하며, 메시지 형태는 제어 메시지, 데이터 메시지, 긴급 메시지를 지원한다. (그림 2)는 XNIF 보드의 구조도이며, 긴급 메시지 관련 내용은 생략한다.

XNIF의 SNI는 MSB(Message Send Buffer), MCR (MSB Control Buffer), SSR(Send Status Register), ACR(Acknowledge Processing Register), NCR(Network COntrol Register), NIR(Node Identifier Register) 자원과 데이터 메시지 전송을 위해서 노드 내 지역 메모리에 있는 데이터를 직접 읽어오는 DCU(DMA Control Unit), SNI와 PCI(Peripheral Component Interconnect) 버스와의 인터페이스를 제공하는 BIU (Bus Interface Unit), MSB 버퍼의 쓰기 및 읽기를 제어하는 BCU(Buffer Control Unit), 전송 오류가 발생한 경우 전송을 재시도 하는 RCU(Retry Control



(그림 2) XNIF 보드 구조
(Fig. 2) XNF Board Architecture

Unit), 메시지를 Xcent-Xet으로 송신 및 응답 송신을 제어하는 MSU(Message Send Unit), Xcent-Net으로 패킷을 송신하는 PSU(Packet Send Unit), Xcent-Net에 연결된 두개의 출력 포트 중재와 연결망을 제어하는 NCU(Network Control Unit)등으로 구성된다. MSB는 두개로 구성되며, 각각의 깊이는 64 바이트 크기의 버퍼로 4이다. 이 버퍼는 메시지 전송 때 사용되는 버퍼로, 마이크로 커널에서는 각 버퍼에 대해서 깊이가 하나인 것처럼 사용되도록 지원된다. MCR은 MSB에 있는 메시지 개수의 상태와 MSB를 제어하는데 사용되는 레지스터로, 각 MSB에 대해 메시지 개수는 헤드(head)/테일(tail) 필드를 두어 관리하며 MSB에 저장되는 메시지 제어를 위해서 쓰기(write)/읽기(read) 필드를 갖는다. SSR은 전송된 메시지의 전송 오류 등을 나타내는 레지스터로, 각 MSB에 대해 패리티(parity) 오류, 순서 오류, 버퍼 충만(full) 오류, 수신 노드 오류 등의 필드를 갖는다. ASR은 타 노드로부터 수신한 메시지에 대한 전송 응답을 회신하는데 사용되는 레지스터이며, APR은 회신된 전송 응답 정보들을 기록하는데 사용된다. NCR은 Xcent-Net을 제어하기 위해서 사용되고, NIR은 노드의 고유 식별자를 갖고 있다. XNIF의 RNI는 MRB(Message Receive Buffer), MRR(MRB control Register), TAR (TDB Address Register), TFR(TDB Flag Register) 자원과 Xcent-Net의 패킷 전송 프로토콜에 맞춰 데이터를 수신하고 패킷 버퍼에 저장하는 PRU(Packet Receive Unit), 패킷 버퍼에서 메시지를 RNI 내부 또는 지역 메모리로 저장하는 MRU(Message Receive Unit)로 구성된다. MRB는 네개로 구성되며, 각각의 깊이는 64 바이트 크기의 버퍼로 4이다. 이 버퍼는 메시지 수신에 사용된다. MRR는 MRB를 제어하는데 사용되는 레지스터로, 각 MRB에 대해 메시지 관리를 위한 필드를 갖는다. 그리고 XNIF 보드에는 없고 데이터 메시지 수신을 위해 지역 메모리에 있는 TDB는 64 바이트 이상의 큰 메시지를 임시적으로 저장하는데 사용되는 버퍼로 1메가 바이트 크기의 8개 버퍼로 구성되며, TAR에 의해 위치를 알 수 있다. TFR는 TDB의 상태를 나타내는 레지스터로, 8개 버퍼에 대한 상태 필드를 갖는다.

앞서 소개한 자원 중에 SNI에 MSB, MCR, SSR과 RNI의 MRB, MRR, TAR, TFR의 인터페이스를 따르면 마이크로 커널은 Xcent-Net상의 제어 및 데이터 메시지를 송수신할 수 있다.

제어 메시지는 최대 64 바이트 크기를 갖는 메시지로, MISIX에서는 전송 송류 처리, 서버 위치 확인 등에서 사용된다. 이 제어 메시지는 수신 노드, 전송 방식, 메시지 길이 등의 전송 정보와 전송할 메시지를 64 바이트 내에 구성하여 MSB에 쓰기만 하면 메시지가 전송된다. 반면에, 데이터 메시지는 64 바이트 이상으로 구성되는 메시지로, MISIX에서는 서버간 서비스 요청 및 정보 전달, 입출력 등에서 사용된다. 이 데이터 메시지는 수신 노드, 전송 방식, 메시지 길이, 전송할 메시지 위치 등의 전송 정보를 구성하여 MSB에 저장되며, 전송될 메시지는 지역 메모리에 구성한다. XNIF에는 데이터 메시지 전송을 위해 DMA 방식을 지원한다.

MISIX에서는 전송 방식과 메시지 형태의 조합으로 여러가지의 전송 형태가 존재한다. 본 논문에서는 구성의 간결화와 내용의 이해를 돕고자 데이터 메시지 형태의 점대점 전송 방식에 초점을 두어 라우터 에뮬레이터의 설계 및 구현 내용을 기술한다.

2.3 MISIX 구조

SPAX에 탑재되는 MISIX는 Chorus 마이크로 커널 v3 r5[13]에 UnixWare 2.0 기능을 서버화한 운영체제(UnixWare/MK)이다. Chorus 마이크로 커널은 모든 노드에서 수행되며, 쓰레드 수행 환경, 가상 메모리 관리, 예외 및 인터럽트 처리, 마이크로 커널간 동기식 및 비동기식 메시지 통신 기능 등을 서버들에게 제공한다. 서버들은 마이크로 커널 인터페이스를 통해서 이러한 기능들을 제공받을 수 있다. 사용자 프로세스는 PN에서 수행되며, PN에서 수행되는 서버로는 프로세스 관리자, 화일 관리자, 스트림 관리자, 데드록 관리자 등 12개가 있다. 그리고 ION에는 SCSI 디스크 관련 서버들이 수행되며, CCN에는 부트, 형상 관리, 콘솔 및 터미널 서버들이 수행된다[14, 15]. MISIX 운영체제를 구성하는 서버들은 서버들의 기능에 맞게 각 노드에 분산되어 존재하게 된다. 이것은 마이크로 커널 기반의 운영체제의 중요한 특징이며, 이 특징으로 시스템의 확장성과 가용성을 높일 수 있다.

마이크로 커널 기반의 운영체제는 포트(port) 기반

의 통신을 한다[1, 13, 16]. Chorus 및 MACH 마이크로 커널에서 포트 기능을 제공한다. 이들 마이크로 커널상의 여러 노드에서 수행되는 서버들은 다른 서버들과 메시지 통신을 하기 위해서, 마이크로 커널에서 제공하는 포트 기반의 메시지 통신을 해야 한다. 서버는 마이크로 커널에서 제공하는 메시지 통신 인터페이스를 사용하면, 위치에 투명한 메시지 통신할 수 있다. Chorus 커널의 포트 기반 메시지 통신은 Ethernet으로 구현되어 있지만, MISIX에서는 Xcent-Net으로 구현되어야 한다. Xcent-Net 기반의 MISIX IPC은 XNIF에서 제공하는 인터페이스들을 따르면 메시지 송수신이 가능하다.

3. XNIF 에뮬레이터 설계

XNIF 에뮬레이터는 XNIF의 SNI와 RNI 부분의 자원과 동작을 C++ 언어로 프로그램하여 마이크로 커널에 추가하였다. 이 에뮬레이터를 이용하여 MISIX IPC는 XNIF 보드에 대한 메시지 송수신 동작을 가상적으로 수행한다. 2장에서 언급했듯이 XNIF는 지정된 메시지 형태에 따라 메시지 송신 및 수신을 독자적으로 담당하는 하드웨어 보드로써, XNIF의 자원과 동작을 소프트웨어로 구현하기 위해서는 다음과 같은 중요한 설계 고려 사항들이 있다.

Issue 1. XNIF 에뮬레이터 구조
Issue 2. XNIF 에뮬레이터 인터페이스
Issue 3. XNIF 메시지 구성
Issue 4. 시험 시스템에서 메시지 전송 방법

〈표 1〉 XNIF 에뮬레이터를 구성하는 쓰레드 종류
〈Table 1〉 Classification of Threads consisting of XNIF Emulator

종류	부분	갯수	기능
Push Thread	SNI	1	MSB에 메시지 저장 및 이동
Transfer Thread	SNI	1	메시지 변환 및 메시지송신
Receive Thread	RNI	1	메시지 변환 및 MRB, TDB에 메시지 저장
Pop Thread	RNI	1	인터럽트 발생 및 메시지 제거 및 이동
Interrupt Thread	커널	4	메시지 수신 인터럽트 처리

또한, XNIF 자원의 초기화 및 동기화 문제와 에뮬레이터를 구성하는 쓰레드 초기화 문제를 언급할 것이다. 다음 절은 각 Issue들을 자세히 설명하여 XNIF 에뮬레이터의 구성 및 설계 내용을 이해하는데 도움이 되고자 한다.

3.1 XNIF 에뮬레이터 구조

에뮬레이터는 크게 XNIF의 SNI와 RNI 부분으로 구성되며, 에뮬레이터 구조 설계는 XNIF의 다음과 같은 수행 특징과 MISIX IPC와의 관계를 고려해야 한다.

- XNIF 보드의 지능화 : 처리기로부터 송신할 메시지를 또는 Xcent-Net으로부터 수신된 메시지를 받으면, 자체 내의 기능으로 스스로 메시지를 Xcent-Net으로 송신하거나 처리기에게 메시지 수신 인터럽트를 발생함
- XNIF 보드의 SNI/RNI간 병렬 처리 : 메시지 송신과 수신 처리를 동시에 수행함
- XNIF 보드의 SNI/RNI내의 병렬 처리 : 임의의 메시지 송신(수신) 처리 중에도 다른 메시지 송신(수신) 요청을 수행함
- MISIX IPC 인터페이스 : 2장에서 기술한 XNIF 보드의 자원 및 기능 중에서 MISIX IPC와 연관되는 자원은 MSB, MCR, SSR, MRB, MRR, TAR, TFR 자원이며, 이들 자원은 MSU, BCU, PRU, MRU 기능으로 제어된다. 나머지 자원과 기능들은 SPAX 하드웨어 환경 및 Xcent-Net 환경에서 필요한 기능임

메시지의 송수신을 스스로 처리하고 처리 과정을 병렬화하기 위해서, 에뮬레이터를 〈표 1〉과 같은 쓰레드들로 구성한다.

Push Thread는 XNIF의 SNI 기능을 수행하는 쓰레드로써, MISIX IPC 송신부에서 메시지를 MSB에 쓰기 동작을 수행하면 해당 MSB의 메시지 양을 고려하여 메시지를 큐잉(queueing)하도록 설계하여, 2장에서 언급했던 MSB 버퍼 깊이가 하나인 것처럼 보이도록 한다. Transfer Thread도 XNIF의 SNI에 속하는 쓰레드다. 이 쓰레드는 Push Thread로 송신될 메시지가 도착하면, 메시지를 시험 시스템에서 전송 가능한

〈표 2〉 XNIF 에뮬레이터 인터페이스
〈Table 2〉 XNIF Emulator Interface

분류	종류	해당 부분	주체	대상
MISIX IPCc	MSBsearch	SNI	MK	MSR
	MSBread	SNI	MK	Push Thread
	MSBwrite	SNI	MK	Push Thread
	Transfer Ack	SNI	Transfer Thread	MK
	Interrupt	RNI	Pop Thread	Interrupt Thread
	MRBread	RNI	MK	MRB
	ReadDone	RNI	MK	Pop Thread
에뮬레이터 내부	SendEnable	SNI	Push Thread	Transfer Thread
	MRBsearch	RNI	Receive Thread	MRR
	TDBsearch	RNI	Receive Thread	TFR
	MRBwrite	RNI	Receive Thread	MRB
	TDBwrite	RNI	Receive Thread	TDB
	WriteDone	RNI	Receive Thread	Pop Thread
	MSBmove	SNI	Push Thread	MSB
	MRBmove	RNI	Pop Thread	MRB

메시지 형태로 변환하고 물리적인 송신과 전송 오류를 관리하도록 설계한다. 한편, Receive Thread는 XNIF의 RNI 기능을 수행하는 쓰레드로서, 수신된 메시지를 변환하여 MRB와 TDB에 메시지를 수신하도록 설계한다. Pop Thread는 MRB에 메시지가 수신되면 메시지 수신 인터럽트를 발생하며, 이때 Interupt Thread는 메시지를 MISIX IPC 수신부에 전달하고 MRB 및 TDB를 반환하도록 Pop Thread에게 알린다. SPAX의 한 노드에 장착되는 처리기 수가 4개이므로 Interrupt Thread를 4개로 구성하고, 나머지 쓰레드는 각 1개씩 구성한다. 각 쓰레드는 마이크로 커널 초기화 시 생성되도록 하며, MISIX IPC 송수신부와 에뮬레이터간 인터페이스 및 에뮬레이터 내부 인터페이스로 동작을 시작한다. 이들 인터페이스는 다음 절에서 기술한다.

3.2 XNIF 에뮬레이터 인터페이스

에뮬레이터의 인터페이스는 크게 MISIX IPC 송수신부간 인터페이스와 에뮬레이터 내부 인터페이스로 나눌 수 있다. 〈표 2〉에서 MISIX IPC 송수신간 인터페이스는 메시지 송신, 송신 완료, 수신 처리에 필요한 7개의 인터페이스 구성되며, 에뮬레이터 내부 인터페이스는 에뮬레이터 수행에 필요한 8개의 인터페이스로 구성한다.

이들 인터페이스들은 함수 또는 메시지 형태로 구현된다. 함수로 구현되는 인터페이스는 MSBsearch, MSSBread, MSBwrite, MRBread, MRBsearch, TDBsearch, MRBwrite, TDBwrite, MSBmove, MRBmove 이며, 메시지 형태로 처리되는 인터페이스는 Transfer Ack, Interrupt, ReadDone, SendEnable, WriteDone 이다. 〈표 2〉의 15가지 인터페이스는 다음 기능을 갖는다.

- MSBsearch: MISIX IPC 송신부에서 메시지를 송신하기 위해서 MSB 버퍼를 선택할 때 사용한다. MSB 버퍼 상태는 MCR의 헤더/테일 필드로 알 수 있다.
- MSBread: MSB의 메시지 송신이 완료되면, MISIX IPC 송신부는 MSB로부터 송신 완료된 메시지를 제거를 요청한다. 요청은 MCR의 쓰기/읽기 필드

를 통해서 이루어진다.

- MSBwrite: 선택된 MSB에 MISIX IPC 송신부는 메시지를 저장하고 MCR의 쓰기/읽기 필드를 통해 메시지 저장을 알린다.
- Transfer Ack: MSB에 저장된 메시지의 송신이 완료되면, MISIX IPC 송신부에 송신 완료를 알린다. 그리고 MISIX IPC 송신부는 송신 완료를 기다린다.
- MRBread: 메시지 수신 인터럽트가 발생하면, MRB에서 수신된 메시지를 읽는다.
- ReadDone: MRBread 발생 후, 마이크로 커널은 메시지 제거 요청을 한다. 요청은 MRR를 통해서 이루어진다.
- SendEnable: MISIX IPC 송신부로부터 MSBwrite가 발생하면, 메시지 송신 시작을 Transfer Thread에게 알린다.
- MRBsearch: Ethernet으로부터 수신된 메시지를 MRB에 보관하기 위해서 MRB의 빈 버퍼를 라운드로빈 방식으로 찾는다.
- TDBsearch: Ethernet으로부터 수신된 메시지 중 데이터 메시지를 보관하기 위해서 TDB의 빈 공간을 찾는다.
- MRBwrite: 선택된 MRB에 수신된 메시지를 쓴다.
- TDBwrite: 선택된 TDB에 수신된 메시지 중 전송 정보를 제외한 내용을 쓴다.
- WriteDone: MRBwrite와 TDBwrite가 완료되면, Pop Thread에게 알린다.
- MSBmove: MISIX IPC 송신부에서 MSBread가 발생하면, 해당 MSB의 메시지 수에 따라 메시지를 큐잉 방식으로 MSB 내부에서 이동시킨다.
- MRBmove: 메시지 수신 인터럽트에서 ReadDone이 발생하면, 해당 MRB 수에 따라 메시지를 큐잉 방식으로 MRB 내부에서 이동시킨다.

3.3 XNIF 메시지 구성

MISIX의 서버에서 전달되는 메시지의 구조는 Chorus IPC에서 지원하는 메시지 구조(이하, Chours 메시지) [1, 13, 16]를 사용한다. Chorus 메시지에는 서버간 통신에 필요한 순수한 내용만을 포함하며, MISIX를 구성하는 서버들은 Chorus 메시지 구조만을 인식하여 동작하도록 되어 있다. 따라서 시스템을 구성하는 노드의 통신 매체에 따라, MISIX IPC 송신부에서는 Chorus 메시지에 통신 매체에 필요한 정보들을 덧붙어서 송신할 최종 메시지를 구축해야 한다. SPAX에서는 Chorus 메시지에 2장에서 언급한 제어 및 데이터 메시지에 따라 Xcent-Net 이용에 필요한 수신 노드, 전송 방식, 메시지 길이, 전송할 메시지 위치 등의 전송 정보를 추가하였다. 덧붙어 Chorus IPC는 포트(port) 기법 기반의 통신 메카니즘이므로, XNIF 메시지 구성에는 Chorus 메시지, 통신 매체 정보, 포트 정보 등으로 구성되도록 하였다.

MISIX IPC 송신부는 XNIF 에뮬레이터 상에서 메시지를 송신하므로 Chorus 메시지를 XNIF 메시지 형태로 구축한다. 따라서 에뮬레이터는 XNIF 메시지를 실제 전송을 위해 다음 절과 같이 메시지 변환 기능을 갖도록 해야 한다.

3.4 시험 시스템에서 메시지 전송 방법

본 연구는 Ethernet 환경의 시스템에서 가상의 Xcent-Net를 이용하는 서버간 통신을 지원하기 때문에, 물리적인 통신에 대한 메시지 변환을 고려해야 한다. 메시지 변환은 XNIF 메시지 송신을 담당하는 Transfer Thread에서 이루어지도록 하여, 서버들에는 메시지 구성의 투명성을 제공한다. 메시지 변환 및 전송은 Chorus 커널 v3 r5에서 제공하는 Ethernet 메시지 구성 및 전송 인터페이스를 사용한다.

MISIX IPC 송신부에서는 서버로부터 전달된 Chorus 메시지를 XNIF 메시지 구성하여 에뮬레이터에 전달하면, 에뮬레이터는 XNIF 메시지에 Ethernet 이용에 필요한 정보를 추가한 Ethernet 메시지로 변환하여 해당 노드에 송신을 한다. 즉, 최종 메시지 형태는 Chorus 메시지에 Xcent-Net 및 Ethernet 정보가 포함된 메시지다. 한편, MISIX IPC 수신부에서는 송신부의 메시지 변환 순서의 역순으로 메시지를 서버에게 전달한다. Ethernet 메시지에서 XNIF 메시지 변환은 에뮬레이터의 Receive Thread가 담당하고, XNIF 메시지로 Chorus 메시지 변환 및 해당 서버에게 전달은 MISIX IPC 수신부가 있다.

3.5 기 타

언급한 4가지 사항 뿐만 아니라, 쓰레드 초기화, XNIF 자원 초기화, 동기화 대상에 대해서 고려해야 한다.

1) 쓰레드 초기화

XNIF 에뮬레이터를 구성하는 5가지의 쓰레드들은 마이크로 커널 초기화시 소멸되지 않는 쓰레드로 생성되어 수행 대기 상태가 되도록 한다. 그리고 이들 쓰레드의 우선 순위는 커널 쓰레드 우선순위 값을 갖도록 한다.

2) XNIF 자원 초기화

XNIF의 많은 하드웨어 자원중 에뮬레이터에서 사용되는 XNIF 자원인 MSB, MCR, SSR, MRB, MRR, TDB, TFR, TAR들을 할당받고 초기화를 한다. 덧붙어, XNIF 메시지 구성에 필요한 자료구조들의 구성과 자료구조 할당 및 반환 함수를 초기화 한다.

3) 동기화 변후

XNIF 에뮬레이터에서는 8가지 다음과 같은 동기화 변수를 고려해야 한다. MSBaccess, TDBaccess, MRBaccess, SSRaccess, MRRaccess 동기화 변수는 메시지의 송수신, 송신 에러 보고 및 조사, 버퍼 반환 등을 위해서 각각 MSB, TDB, MRB, SSR, MRR 자원 접근시 필요하다. var 동기화 변수는 에뮬레이터 내부의 공유 변수 관리를 위해서 사용하고, want 동기화 변수는 MSB, MRB, TDB 버퍼 사용 요청에 사용된다. mail 동기화 변수는 쓰레드간 인터페이스에 필요한 정보 구성에 사용된다. 이들 변수는 다음 장의 쓰레드 알고리즘에 사용된다.

4. XNIF 에뮬레이터의 구현

앞 장의 주요 설계 내용을 기반으로 UnixWare/MK에서 수행되는 에뮬레이터를 C++로 구현하였다.

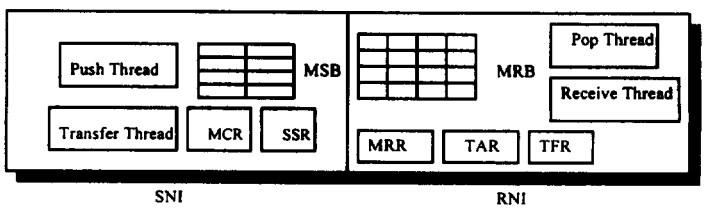
4.1 SNI 부분

1) SNI 부분 구조

XNIF의 SNI 인터페이스 중 마이크로 커널의 MISIX IPC 송신부와 연관되는 인터페이스는 MSB, MCR, SSR 자원을 통해서 이루어지며, 이들 인터페이스에 따른 동작은 XNIF 메시지를 큐잉하는 동작과 메시지를 원격 노드에 전송하고, 전송 결과를 MISIX IPC 송신부에 알리는 동작이다. 따라서 (그림 3)의 SNI 부분과 같이 SNI 에뮬레이터 구조는 MSB, MCR, SSR 자원과 두가지 동작을 하는 2개의 쓰레드로 구성된다. 첫번째 쓰레드는 MISIX IPC 송시누가 삽입하거나 제거하는 XNIF 메시지에 대해 MSB 상태에 따라 큐잉을 하며 두번째 쓰레드는 전송할 메시지가 있으면 메시지 변환을 하여 메시지를 원격 노드에 전송을 한다. 전자의 경우는 Push Thread이고 (그림 2)의 BCU에 해당하며, 후자는 Transfer Thread이고 MSU에 해당된다. 이들 쓰레드는 마이크로 커널 초기화 시 소멸되지 않은 쓰레드로 생성되어 수행 대기 상태로 존재하고, 자원은 에뮬레이터 수행 시작시 할당하여 초기화한다.

2) 쓰레드 기능

(그림 4)는 에뮬레이터에서 MSB 버퍼의 쓰기 및 읽기를 제어하는 Push Thread의 슈도우(pseudo) 코드이다. 여기에서 두가지 중요한 구현 내용이 있다. 첫째, MISIX IPC 송신부에서 XNIF 메시지를 MSB에 쓸 때, MSB 버퍼 깊이가 하나인 것처럼 보이도록 한다. 즉, 자동적인 메시지 큐잉 방식을 제공해야 한다. 코드에서 메시지 큐잉 방식은 에뮬레이터 내부 인터페이스인 MSBmove로 구현한다. MSBmove는 MSB에 이미 저장된 메시지 수에 따라 해당 위치에 XNIF



(그림 3) XNIF 에뮬레이터 구조
(Fig. 3) XNIF Emulator Architecture

메시지를 옮겨놓는다. 그리고 메시지 송신 후, 메시지 제거 때도 MSBmove를 사용하여 메시지를 이동시킨다. Push Thread 수행 전에는 반드시 MSBaccess 동기화 변수로 임계 영역(critical section) 사용 허가를 획득해야 한다. 둘째, XNIF 메시지 전송을 시작 시키는 시점이다. 즉, XNIF 메시지가 MSB에 저장될 때마다 Transfer Thread에게 전송을 의뢰하는 것은 아니다. MSBmove후 메시지가 해당 MSB의 최하위 버퍼에 이동된 경우에 전송을 의뢰한다. 또한, 전송 완료 후 MISIX IPC 송신부의 메시지 제거 요청 후에도 마찬가지다.

```
void Push_Thread()
input : NONE
output : NONE
{
    while(1) {
      wait for notice from MISIX IPC sender;
      get the MSBaccess synchronization variable;
      if (write-flag of MSB in MCR is on) { /* have wrote on MSB? */
                increase the number of messages;
                clear write-flag of MSB;
                MSBmove;
                if (the number of message == 1)
                      give notice to Transfer Thread;
      } else if (read-flag of MSB in MCR is on) {
                if (there are some errors in a message transmission)
                      clear SSR;
                decrease the number of messages;
                clear read-flag of MSB;
                if (there are messages in MSB) {
                      MSBmove;
                      give notice to Transfer Thread;
                } else
                      MSBmove;
      }
      release the MSBaccess synchronization variable;
    }
}
```

(그림 4) Push Thread 슈도우 코드
(Fig. 4) Pseudo Code of Push Thread

```
void Transfer_Thread()
input : NONE
output : NONE
{
   while(1) {
      wait for notice from Push Thread;
      decision of a MSB side with round-robin method;
      read the message from MSB selected at previous step;
      DoRealTransfer(); /* virtually transfer a message to Xcent-Net */
      after transfer of message, if there is some error of transfer
      write error information on SSR;
      give a transfer acknowledgement to MISIX IPC sender;
   }
}
```

(그림 5) Transfer Thread 슈도우 코드
(Fig. 5) Pseudo Code of Transfer Tnread

(그림 5)는 시험 시스템에서 XNIF 메시지 전송을 담당하는 Transfer Thread의 슈도우 코드이다. 이 쓰레드는 Push Thread에 의해 메시지 전송이 시작하며, Push Thread와 병렬 수행된다. MSB에 저장되는 메시지는 라운드 로빈(round robin) 방식으로 처리된다. 선택된 XNIF 메시지는 메시지 변환 후 Ethernet으로 전송하여 가상적인 Xcent-Net 전송이 이루어지도록 한다. 메시지 변환 및 Ethernet 전송은 다음 항목에서 설명한다. 전송이 완료되면 전송 오류가 있는지 조사하여 오류 정보를 SSR에 기록하고, MISIX IPC 송신부에게 전송 완료를 통보한다. 전송 완료를 통보 받은 MISIX IPC의 송신부는 XNIF 메시지를 MSB에서 제거하도록 Push Thread에게 알린다.

3) 메시지 변환

메시지 변환은 Chorus 커널 v3 r5 IPC 인터페이스[13]를 이용한다. Transfer Thread는 MSB에서 읽은 메시지 내용(이하, Xnifhead)와 지역 메모리에 있는 메시지 내용(이하, Xnifbody)로 Ethernet 메시지를 구축한다. 변환 방법으로는 Xnifhead와 Xnifbody를 전송할 데이터로 간주하고 Ethernet 이용 정보를 추가하도록 하였다.

```
int DoRealTransfer()
input : Xnifhead pointer(hd)
output : transfer status
{
      get destination node and message size from hd;
      get address of Xnifbody from hd;
      /*Next, use CHORUS Kernel v3 r5 IPC framework interface to build
      and transfer ethernet message
      (getIpcSession(), outMsgAllocate(), ipcSendMsgHeaderPush(), push()) */
      build ethernet message with Xnifhead and Xnifbody;
      transfer a ethernet message;
      return transfer status;
}
```

(그림 6) 메시지 변환 및 전송 슈도우 코드
(Fig. 6) Pseudo Code of Message Conversion and Transmission

메시지 변환에서 중요한 사항은 Ethernet 메시지 구성 시 XNIF 메시지 형태를 그대로 유지하여야 할 것이다. 왜냐하면, MISIX IPC 수신부가 XNIF 메시지 형태만 이해할 수 있기 때문이다. 본 논문은 (그림 6)의 DoRealTransfer 함수에서 Xcent-Net상의 점대점(point-to-point) 통신에 대한 변환 절차만을 기술한다.

4) 에뮬레이터상에서 메시지 전송 과정

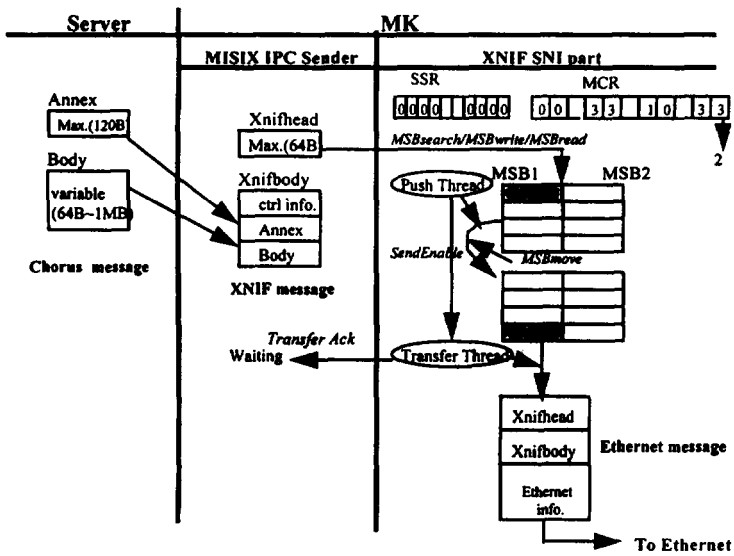
(그림 7)은 에뮬레이터에서 첫번째 메시지가 전송하는 모습이다. 서버에서 사용하는 Chorus 메시지는 Annex와 Body로 구성된다. 마이크로 커널의 MISIX IPC 송신부에서 Xcent-Net에 대한 전송 정보인 Xnifhead와 포트 정보 ((그림 7)의 ctrl info) 및 Chorus 메시지로 구성한 Xnifbody로 XNIF 메시지를 만든다. Xnifhead는 빈 MSB 버퍼에 저장하며, Xnifbody는 Xnifhead가 알고 있는 지역 메모리에 둔다. (그림 7)에서 이태릭체는 앞장에서 정의한 인터페이스를 나타낸다. MISIX IPC 송신부가 XNIF 메시지를 MSB1에 기록하고 MCR의 쓰기/읽기 필드를 1/0로 설정하여 MSB1에 메시지 저장을 알린다(MSBsearch, MSBwrite). Push Thread는 MCR를 통해서 송신할 메시지가 있다는 것을 인식하고 메시지를 최하위의 버퍼로 이동시키고 MSB1의 메시지 수 상태를 MCR의 헤드/테일 필드에 기록한다(MSBmove). 그리고 Transfer Thread에 SendEnable를 보내어 전송을 시작 시킨다.

SendEnable를 받은 Transfer Thread는 최하위의 버퍼에서 메시지를 읽어 XNIF 메시지에 Ethernet 사용 정보를 추가하여 Ethernet 메시지를 만들어 전송을 완료하고, Transfer Ack를 MISIX IPC 송신부에 보낸다. Transfer Ack를 받은 MISIX IPC 송신부는 메시지를 제거하기 위해서 MCR의 쓰기/읽기 필드를 0/1로 설정하여 Push Thread에 요청한다(MSBread). 메시지 제거 요청 받은 Push Thread는 MSB1에서 다른 메시지를 아래로 이동시켜 메시지를 제거하고 다른 메시지 전송을 계속하도록 한다.

4.3 RNI 부분

1) RNI 부분 구조

XNIF의 RNI 자원중 MISIX IPC 수신부와 연관되는 인터페이스는 MRB, MRR, TAR, TFR 자원을 통해서 이루어지며, 이들 인터페이스에 따른 동작은 수신된 메시지를 수신 버퍼에 관리하는 동작과 처리기에서 메시지 수신 인터럽트를 발생하는 동작이다. 따라서 (그림 3)의 RNI 부분과 같이 MRB, MRR, TAR, TFR과 2개의 쓰레드로 구성한다. 첫번째 쓰레드는 Ethernet으로부터 수신된 메시지를 역 변환하고 MRB 및 TDB의 상태를 조사하여 XNIF 메시지를 수신 버퍼에 기록하는 쓰레드와 두번째 쓰레드는 수신된 메시지가 존재하면 해당 처리기에 메시지 수신 인



(그림 7) 에뮬레이터상에서 메시지 송신 과정
(Fig. 7) Message Send Procedure in Emulator

터럽트를 발생시키고, 인터럽트 핸들러가 메시지 제거를 요청하면 수신 버퍼에서 메시지의 제거 및 이동을 담당하는 쓰레드이다. 전자는 Receive Thread이고 (그림 2)의 PRU에 해당되고, 후자는 Pop Thread이고 MRU에 해당된다. 그리고 메시지 수신 인터럽트 발생은 소프트웨어 방식으로 처리하기 때문에, 처리기 수 만큼의 인터럽트 쓰레드를 구현하여 인터럽트를 처리하도록 한다.

2) 쓰레드 기능

(그림 8)은 시험 시스템에서 Ethernet 메시지를 수신하여 XNIF의 RNI 버퍼에 메시지 저장을 담당하는 Receive Thread 슈도우 코드이다. 이 쓰레드는 Ethernet 메시지를 입력받아 동작을 한다. 송신 때 Xnifhead와 Xnifbody로 구성한 XNIF 메시지를 Ethernet 메시지에서 쉽게 구할 수 있다. Xnifhead는 MRB 버퍼에 저장되는데 라운드로빈 방식으로 MRB의 버퍼를 할당한다. 그리고 Xnifbody는 빈 TDB 버퍼에 저장한다. 이때, 송신 시 기록한 Xnifhead 내용 중 Xnifboody를 가리키는 정보를 변경해야 한다. 왜냐하면, Xnifbody가 수신 노드의 TDB 공간에 할당되기 때문이다. 이들 버퍼가 부족하면, Pop Thread에 의해서 반환될때까지 기다려야 한다. Ethernet에서 에뮬레이터의 수신 버퍼로 수신이 완료되면, Pop Thread에게 알려 메시지 수신 인터럽트를 처리하도록 한다. 그림에서 이태릭 문자는 사용 가능한 MRB와 TDB 버퍼를 찾을 때 사용되는 MRBaccess와 TDBaccess, 빈 버퍼가 없어 버퍼 할당을 요구할 때 사용되는 want, 그리고 Pop Thread에게 메시지 수신을 알리기 위해 필요한 자료구조 할당 때 사용되는 mail 동기화 변수다.

(그림 9)은 메시지 수신 인터럽트 발생과 MRB 버퍼 제어를 담당하는 Pop Thread 슈도우 코드다. 처음에는 Receive Thread에 의해서만 이 쓰레드가 수행되지만, 메시지 수신 인터럽트 발생 후에는 Receive Thread나 Interrupt Thread에 의해 수행될 수 있다. 수행을 시작하면, 먼저 다른 쓰레드에서 전달된 수행 정보를 반환한다. MRB에 저장된 XNIF 메시지가 MRB의 최상위 버퍼에 있다면 메시지 수신 인터럽트를 발생시킨다. 메시지 수신 인터럽트 처리에서 XNIF 메시지를 MRB 및 TDB에서 복사하고, Pop

```
void Receive_Thread()
input : Ehthernet message (msg)
output : NONE
{
    build XNIF message(Xnifhead, Xnifbody) with msg;
    ASSERT(It has Xnifhead);
    get MRBaccess synchronization variable;
    search an empty MRB buffer by Round-Robin method;
    release MRBaccess synchronization variable;
    if (there is no MRB buffer) {
        get want synchronization variable;
        request a MRB buffer to be used;
        release want synchronization variable;
        wait for an empty buffer released by Pop Thread;
    }
    if (need TDB buffer) {
        get TDBaccess synchronization variable;
        search an empty TDB buffer;
        release TDBaccess synchronization variable;
        if (there is no empty TDB buffer) {
            get want synchronization variable;
            request a TDB buffer to be used;
            release want synchronization variable;
            wait for an empty buffer released by Pop Thread;
        }
        initialize the pointer of allocated TDB buffer in Xnifhead;
        copy Xnifbody contents into allocated TDB buffer;
    }
    copy Xnifhead contents into allocated MRB buffer;
    get mail synchronization variable;
    allocate a mail data structure and send it to Pop Thread;
    release mail synchronization variable;
    free msg;
}
```

(그림 8) Receive Thread 슈도우 코드
(Fig. 8) Pseudo Code of Receive Thread

```
void Pop_Thread()
input : none
output : none
{
  while(1) {
    wait for mail from Receive Thread or Interrupt Thread;
    get mail synchronization variable;
    free mail data structure;
    release mail synchronization variable;
    if (ReadDone from Interrupt Thread) { /* wrote on MRR */
        MRBread;
        get MRRaccess synchronization variable;
        clear ReadDone at MRR;
        release MRRaccess synchronization variable;
    }
    if (there is a message to be interrupted)
        give WriteDone notice including MRB information
        to Interrupt Thread;
  }
}
```

(그림 9) Pop Thread 슈도우 코드
(Fig. 9) Pseudo Code of Pop Thread

```
void Interrput_Thread()
input : none
output none
{
   while(1) {
      wait for notice included MRB information  from Pop Thread;
      allocate memory for copying the contents of MRB and TDB;
      copy MRB and TDB contents;
      get MRRaccess synchronization variable;
      write ReadDone;
      release MRRaccess synchronization variable;
      get mail synchronization variable;
      to free both MRB and TDB, give notice to Pop Thread;
      release mail synchronization variable;
      give an  XNIF message to MISIX IPC receiver;
   }
}
```

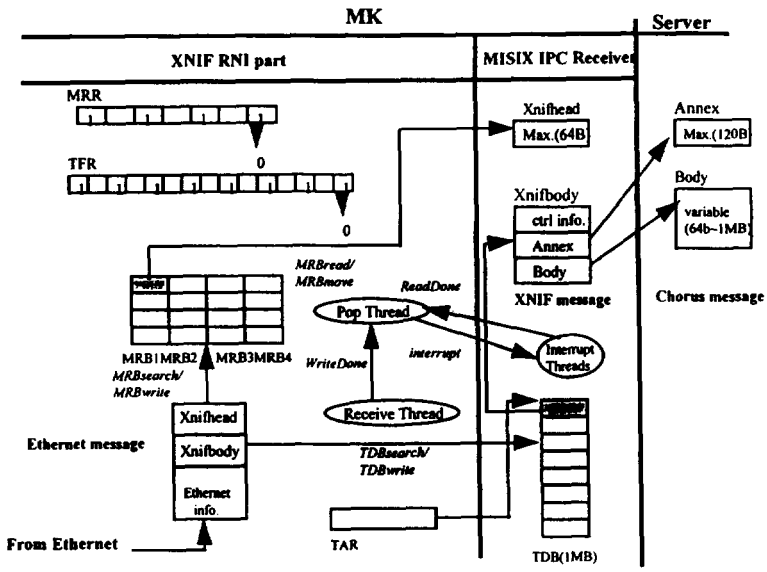
(그림 10) Interrupt Thread 슈도우 코드
(Fig. 10) Pseudo Code of Interrupt Thread

Thread에게 메시지 제거를 요구한다. 메시지 제거는 먼저, MRB의 내용으로 해당 TDB 버퍼를 찾아 TDB 버퍼를 반환하고, 마지막으로 MRB 버퍼를 반환한다. 그리고 MRB에 저장된 메시지를 이동시켜 수신된 메시지가 있으면 다른 인터럽트를 발생시킨다.

(그림 10)은 Interrupt Thread 슈도우 코드다. 에뮬레이터에서 메시지 수신 인터럽트를 소프트웨어 방법으로 처리하기 때문에, 기존의 Chorus 커널 v3 r5 인터럽트 처리 과정과 다른 방법으로 처리하였다. 이 쓰레드는 Pop Thread에 의해서 수행된다. 메시지 수신 인터럽트는 하나의 쓰레드에게 발생되기 때문에, 메시지를 읽기 위해서 MRB에 대한 동기화 변수는 필요가 없다. 메시지를 MRB와 TDB에서 읽은 후, 다음 메시지 수신을 위해서 Pop Thread에게 메시지 제거를 요구해야 한다. 읽은 메시지는 마이크로 커널의 MISIX IPC 수신부에게 전달되고, 마이크로 커널 수신부도 Chorus 메시지 형태로 재구성하여 해당 서버에게 전달된다.

3) 에뮬레이터 상에서 메시지 수신 과정

(그림 11)은 에뮬레이터 상에서 첫번째 메시지가 수신되는 모습이다. XNIF의 하드웨어 자원인 MRB, TAR, MRR, TFR를 자료구조로 표현하고 MRB와 TDB를 관리하는 하드웨어 동작을 두개의 쓰레드로 표현한다. Interrupt Thread는 마이크로 커널의 기능이지만, 에뮬레이터의 수신 동작을 잘 이해할 수 있도록 이 그림에 포함시킨다. 이 그림은 처음 메시지가 수신되었을 때의 모습을 나타내고, 이태릭체는 앞장에서 정의한 인터페이스를 나타낸다. Ethernet 메



(그림 11) 에뮬레이터상에서 메시지 수신 과정
(Fig. 11) Message Receive Procedure in Emulator

시지를 수신한 Receive Thread는 라운드로빈 방식으로 MRB를 선택하고 빈 TDB를 찾는다(MRBsearch, TDBsearch). Ethernet 메시지 중 Xnifhead는 MRB에 저장하고 Xnifbody는 TDB에 저장한다(MRBwrite, TDBwrite). 이 그림에서는 첫번째 메시지가 수신되었기 때문에, 최상위 버퍼에 메시지를 저장하고 메시지 저장을 Pop Thread에게 알린다(WriteDone). Pop Thread는 메시지가 저장된 MRB에 대해 Interrupt Thread에게 메시지 수신 인터럽트를 통보한다(Interrupt). Interrupt Thread는 수신한 메시지를 읽고 다음 메시지 수신을 우해서 메시지 제거를 요청하며(ReadDone), XNIF 메시지를 MISIX IPC 수신부에 전달한다. 메시지 제거 요청을 받은 Pop Thread는 해당 MRB의 최상위 버퍼 내용 및 해당 TDB 버퍼 내용을 제거하고, 해당 MRB 버퍼 내용을 윗 방향으로 한 단계씩 이동(MRBmove)시켜 수신을 계속한다.

5. 검 증

앞 절에서 기술한 XNIF 보드에 대한 에뮬레이터 기능들은 MISIX 운영체제의 마이크로 커널 부분에 삽입하여 기존의 MISIX 커널 컴파일 환경 및 방법으로 커널을 제작하였다. 개발된 에뮬에니터 검증은 두 단계를 통해서 이루어졌다. 첫번째 단계로, 제작된 커널은 두 대의 Compaq Proliant 4000 시스템 각각의 부트 화일 시스템에 두고, 정해진 방법으로 두 시험 시스템 부팅하였다. 시스템 부팅에서 임의의 한 시스템은 마스터(master) 시스템이고 나머지 시스템은 슬래이브(slave) 시스템이 된다. 2장에서 기술 했듯이 MISIX의 시스템 서버들이 각 기능에 맞게 분산되어 있으므로, 슬래이브에 존재하는 서버들은 자신의 포트 정보들을 마스터 시스템에게 알려야 한다. 알려진 정보는 다른 시스템에서 해당 서버를 호출할 때 사용된다. 이러한 과정에서 슬래이브의 마이크로 커널은 MISIX IPC를 이용하여 원격에 있는 마스터 시스템과 통신을 한다. 서버들의 포트 정보 등록 뿐만 아니라, 커널 초기화 시 필요한 원격 시스템의 자원(예, 파일 시스템 등)들의 접근에도 MISIX IPC가 사용된다. 이러한 가정들을 거쳐 다중 사용자 모드까지, 마스트 시스템은 원격 전송을 92번, 슬래이브 시스템은 86번을 에뮬레이터 환경에서 수행하여 커널 부팅이 성공적으로 되었다.

두번째 단계로는, 성공적인 커널 부팅 완료 후, 원격 시스템간 통신 명령어를 이용하여 다양한 작업을 수행하였다. 원격 시스템간 통신 명령어는 on 명령어(구문:on 시스템주소 명령어)를 사용하였다. 기본적으로, on 명령어는 수행될 명령어를 지정한 시스템에 프로세스 이전(migration)시킨다. MISIX 운영체제에서 프로세스 이전은 0x2000 바이트의 메시지 내용이 원격 시스템에 전송되므로서 이루어진다. 따라서, on 명령어를 사용하여 ls, ps, who 등의 유닉스 명령어들을 동시에 수행시켜 에뮬레이터에 빈번한 메시지 송수신이 발생되도록 하였다. 이러한 작업들도 커널 패닉 등의 오류없이 완료되었다.

6. 결 론

본 논문은 SPAX 시스템의 Xcent-Net 내부 네트웍 라우터인 XNIF 하드웨어 동작에 대한 에뮬레이션 프로그램의 설계 및 구현을 제시하였다. 이 라우터 에뮬레이터는 Ethernet으로 연결된 두 대의 Compaq Proliant 4000에서 동작되는 것으로써, 설계 및 구현 때 두 가지에 중점을 두었다. 첫째, XNIF 동작의 기능에 맞는 구조를 갖도록 하였고 둘째, Ethernet를 이용하여 메시지가 전송되도록 메시지 변환을 고려하였다. 에뮬레이터는 MISIX 운영체제의 마이크로 커널에 포함되며, MISIX IPC 송수신부는 에뮬레이터에서 제공하는 7개의 인터페이스를 이용하면 메시지 송수신을 할 수 있었다. 에뮬레이터를 포함하는 운영체제는 Ethernet으로 연결된 두 대의 시험 시스템에서 부팅 과정과 원격 시스템간의 통신 명령어를 사용한 다양한 작업을 통해서 검증되었다. 에뮬레이터 환경에서 구현된 MISIX IPC는 SPAX 시스템에서 쉽게 재사용할 수 있으며 MISIX IPC 기능을 이용하는 서버의 메시지 처리부도 구현이 가능하게 되어, MISIX 개발 일정에 큰 도움을 주었다.

향후 계획으로는 대량의 데이터 메시지의 효율적인 송수신 방법에 대한 성능을 측정할 것이다. OLPT (On-Line Transaction Processing) 시장을 목표로 하는 SPAX 시스템은 메시지 전달 기반의 시스템으로 효율적인 대량의 메시지 송수신은 시스템의 성능에 직접적인 영향을 주기 때문이다.

참 고 문 헌

[1] Chorus systems, "Overview of he Chorus Distributed Operating System," Proceeding of the USENIX workshop on Micro-Kernels and Other Kernel Architectures, pp. 39-69, April 1992.

[2] 박승규, "대용량 서버용 병렬처리 컴퓨터 구조," 대한전자공학회 컴퓨터기술지, 제8권, 제1호, 1994년 12월.

[3] J. Udell, "The great OS debate," BYTE, pp. 117-168, January 1994.

[4] Michel Gien, "Micro-kernel architecture-Key to modern system design," Unix Review, November 1990.

[5] Michel Gien and Lori Grob, "Micro-kernel based operating systems:Moving UNIX onto modern system architetures," Proceeding of UniForum'92 Conference, San Francisco, CA, January 1992.

[6] OSF RI, "OSF/1 MK-AD:A standards-compliant, distributed operating system for loosely coupled, massively parallel supercompters," OSF Research Institute Microkernel Investigations, May 1992.

[7] M. Rozier, et. al., "Chorus distributed oprating systems," Computing Systems Journal, vol. 1, no. 4, pp. 305-370, December 1988.

[8] Bill Bryant, Design of AD 2, a Distributed UNIX Operating System. Open Software Foundation Research Institute, April 1995.

[9] M. Accetta, et. al., "MACH:A new kernel foundation for UNIX development," Summer Conference Preceedings of 1986 USENIX, 1986.

[10] Y. W. Kim, S. W. Oh, and J. W. Park, "Design issues and system architecture of TICOMIV, A highly paprallel commercial computer," The 3rd Euromicro Workshop Parallel & Distributed Processing, pp. 219-226, January 1995.

[11] 김해진, 임기욱, "MISIX:고속병렬컴퓨터(주전산기 IV) 병렬 운영체제," UNIEXPO'95 논문지, pp. 45-49, 1995년 11월.

[12] 모상만, 신상석, 윤석한, 임기욱, "소속병렬컴퓨터(SPAX)에서 효율적인 메시지 전달을 위한 메시지 전송 기법," 대한전자공학회 논문집, pp. 1299-1304, 제18권, 제1호, 1995년 6월.

[13] Chorus Systems, "Chorus kernel v3 r5 specification and interface," CS/TR-91-69.2, 1991.

[14] 정낙주, 김주만, 김해진, 임기욱, "고속병렬컴퓨터를 위한 ION 운영체제의 구조 및 설계," 한국정보과학회 추계학술발표논문집(B), pp. 965-968, 제22권, 제2호, 1995년 10월.

[15] 안선희, 김주만, 김해진, 임기욱, "고속병렬컴퓨터의 통신 접속 노드를 위한 마이크로 널 기반 운영체제 구조 설계," 한국정보과학회 추계학술발표논문집(B), pp. 965-968, 제22권, 제2호, 1995년 10월.

[16] 정성인, 최효진, 김해진, 임기욱, "고속병렬컴퓨터에서 포트를 이용한 시스템 서버 통신기법," 한국정보과학회 추계학술발표논문집(B), pp. 961-964, 제22권, 제2호, 1995년 10월.

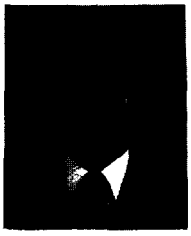
정 성 인

1987년 부산대학교 전산통계학과 졸업(학사)

1989년 부산대학교 대학원 전산학과(이학석사)

1989년~현재 한국전자통신연구원 선임연구원

관심분야:병렬 운영체제, 멀티미디어 시스템, 분산 처리



이 재 경

1981년 경북대학교 전자공학과(공학사)

1983년~1984년 (주)금성반도체

1985년~현재 한국전자통신연구원 책임연구원(컴퓨터연구단 시스템연구부)

관심분야:병렬 운영체제, 병렬 프로그래밍, 컴퓨터 통신

김 해 진

1983년 경북대학교 컴퓨터공학과(학사)
1995년 충남대학교 대학원 전산학과(석사)
1992년 정보처리기술사(전자계산조직응용)
1983년~현재 한국전자통신연구원, 책임연구원
관심분야:운영체제, 병렬 처리, 실시간 처리, 고장 감내

임 기 욱

1977년 인하대학교 공과대학 전자공학과 졸업(학사)
1986년 한양대학교 대학원 전자계산학과(공학석사)
1994년 인하대학교 대학원 전자계산학과(공학박사)
1977년~1983년 한국전자기술연구소 선임연구원
1983년~1988년 한국전자통신연구소 시스템 소프트웨어 연구실장
1988년~1989년 미 캘리포니아 주립대학(Irvine) 방문연구원
1989년~1997년 한국전자통신연구원 시스템연구부장
1997년~현재 정보통신연구관리단 정보기술전문위원
관심분야:소프트웨어 아키텍처, 실시간 데이타베이스 시스템, 컴퓨터 시스템 구조