다목적 클러스터링 시스템을 위한 고속 메시징 계층 구현

박 준 희[†] · 문 경 덕[††] · 김 태 근[†††] · 조 기 환[††††]

요 약

고속의 네트워크로 연결된 클러스터링 시스템은 네트워크 속도를 응용 수준에서도 가시화하기 위해 고속의 메시징 계층을 갖아야 한다. 고속의 메시징 계층은 응용 프로그램의 메시지를 직접 네트워크 카드로 전송할 수 있는 방법을 갖아야 하며, 응용 도메인에 따른 다양한 적용방법을 제공해야 한다. 본 논문에서는 다목적용 클러스터링 서버 시스템인 CROWN (Clustering Resources On Workstations' Network)을 간략히 소개하고, CROWN의 고속 통신 프리미티브로서 구현된 CLCP (CROWN Lean Communication Primitives)의 구현 내용에 대해서 설명한다. CLCP는 Myrinet카드 제어를 위한 펌웨어(Firmware)와 디바이스 드라이버, 사용자 라이브러리로 구성되며, 클러스터링 시스템의 다양한 응용 도메인을 지원하기 위해 폴링과 인터럽트 방식을 모두 지원한다. CLCP는 폴링 기반의 송수신을 할 경우, 8Bytes 메시지를 주고 받을 때 262 μs의 응답시간을 보였으며, 1MBytes의 메시지를 송수신할 때 442Mbps의 대역폭을 제공한다.

Implementation of High Performance Messaging Layer for Multi-purpose Clustering System

Jun-Hee Park[†] · Kyeong-Deok Moon[††] · Tae-Geun Kim[†††] · Gihwan Cho[††††]

ABSTRACT

High speed messaging layer for application's feeling of low level network performance is needed by Clustering System based on high speed network fabrics. It should have the mechanism to directly pass messages between network card and application space, and provide flexible affodabilities for many diverse applications. In this paper, CROWN (Clustering Resources On Workstations' Network) which is designed and implemented for multi-purpose clustering system will be introduced briefly, and CLCP (CROWN Lean Communication Primitives) which is the high speed messaging layer for CROWN will be followed. CLCP consists of a firmware for controlling Myrinet card, device driver, and user libraries. CLCP supports various application domains as a result of pooling and interrupt receive mechanism. In case of polling based receive, 8 bytes short message, and no other process, CLCP has 262 micro-second response time between two nodes, and 1M bytes large message, it shows 442Mbps bandwidth.

1. 서 론

분산 시스템과 병렬 시스템의 중간적인 위치로서 클러스터링 시스템에 대한 관심이 크게 증가되면서 버클리(Berkeley) 대학에서 연구된 NOW(Networks Of Workstations)[18]프로젝트는 워크스테이션급 컴퓨터를 고속의 네트워크로 연결시켜 메인 프레임급 컴퓨터의 성능을 제공하므로써, 가격대 성능비에 있어서 일반 고성능 서버 컴퓨터보다 우수한 시스템을 개발하고자 시작

† 정 회 원 : 한국전자통신연구원 연구원
†† 정 회 원 : 한국전자통신연구원 선임연구원
††† 정 회 원 : (주)DTVRO 대표이사
†††† 정 회 원 : 전북대학교 교수

논문접수 : 1999년 5월 19일, 심사완료 : 1999년 12월 2일

된 클러스터링 시스템 연구 프로젝트이다. 이밖에도 프린스턴(Princeton) 대학의 SHRIMP(Scalable, High-Performance, Really Inexpensive Multi-Processor)[12], CSAG(Current Systems Architecture Group)의 HPVM (High Performance Virtual Machines)[5, 16], 일본의 RWC(Real World Computing) PC 클러스터[17], 이스라엘 히브류(Hebrew) 대학의 MOSIX(Multicomputer Operating System for UNIX)[19], 프랑스 LHPC(Laboratory for High Performance Computing)의 High Speed Networks and Parallel Applications Project[7] 등 많은 대학과 연구기관에 의해 클러스터링 시스템이 연구되고 있다. 이와같은 연구들의 목적은 클러스터링 시스템을 고성능의 컴퓨팅 서버로 활용기 위함이 대부분이며, 고속 통신 프리미티브 기술 역시 병렬 응용을 위해서 고안된 것이 대부분이다. 그러나, 본 연구팀이 개발한 CROWN (Clustering Resources On Workstations' Network)[1-3, 15]은 병렬 컴퓨팅 서버 뿐만 아니라 멀티미디어 서버로서 클러스터링 시스템을 활용하기 위한 시스템이다.

클러스터링 시스템에서 갖추어야할 성분은 여러가지가 있다 그 중 고속통신 프리미티브는 중요한 기반 기술의 하나이다. 상기한 모든 프로젝트들에서도 각기 자기 시스템에 적합한 독자적인 고속 통신 기술을 보유하고 있다. 버클리의 AM(Active Messages)[4, 10, 11, 14, 18], 프린스턴의 VMMC(Virtual Memory Mapped Communication)[12, 13], 일리노이의 FM(Fast Messages)[5, 16], RWCP의 PM[17], LHPC의 BIP(Basic Interface for Parallelism)[7], 코넬(Cornell) 대학의 U-Net(User-level Network)[8] 등이 그것이다. CLCP(CROWN Lean Communication Primitives)[1-3, 15]는 CROWN 시스템의 고속 통신 메커니즘으로서 CROWN 시스템의 두개의 응용분야인 멀티미디어 서버(VOD 서버-MovieRo)와 병렬 컴퓨팅 환경(EPVM)을 지원하기 위해서 설계 및 개발되었다. CLCP는 CROWN 시스템의 각 노드들을 연결하는 초 경량 통신 프리미티브로서, 고속 통신 카드(Myrinet [9])가 갖는 본래의 성능(Raw Performance)을 최대한 통신 어플리케이션(MovieRo, EPVM)에게 까지 전달할 수 있도록 설계 및 구현되었다.

2. 관련연구

Berkeley의 NOW프로젝트 이후 SAN(System Area Network)에 대한 연구가 많은 대학과 연구 기관에 의해서 진행되었고 많은 고속 통신 프리미티브들이 개발되었다. 본 장에서는 AM, FM, LFC, BIP 등에 대해서 알아본다.

AM(Active Messages)[4]은 Berkeley의 NOW 프로젝트에서 개발한 통신 소프트웨어로서 현재 AM-II를 개발하였다. AM-II는 기존의 Active Messages 시스템에 Virtual Network이라는 개념의 확장된 네트워킹 기술을 추가하고 이에 대응되는 API를 제공한다. Virtual Network은 high throughput, reliable message delivery, simple yet powerful error model의 기능을 지원한다. 이를 위해서 Endpoint scheduling, Flow Control, Timer Management, 그리고 Error Handling 기술이 구현되었다.

FM(Fast Messages)[5]은 Illnois에서 개발된 소프트웨어로서 현재 FM-II까지 개발된 상태이다. FM-II는 기존의 FM에서 신뢰성 있는 flow control을 구현하였음에도 불구하고 네트워크의 능력을 사용자에게 충분히 전달하지 못하고 있다는 판단에 근거하여 stream abstraction 개념을 이용하여 주고 받는 대상을 메모리의 임의의 영역에서 byte stream으로 변환함으로써 효율적인 API를 제공함을 목표로 하고 있다. 이를 위해 FM-II에서 구현된 기술은 gather-scatter, layer interleaving, receiver flow control이다.

LFC(Link-level Flow Control)[6]는 병렬 컴퓨팅용 통신 소프트웨어 개발자에게 신뢰성 있는 일대일(point-to-point)과 멀티캐스트 패킷기반 통신 환경을 링크 수준에서 제공함을 목표로 하고 있다. 이를 위해서 LFC는 NI-level credit-based flow control, NI-level multicast mechanism 기술을 지원한다.

BIP(Basic Interface for Parallelism) [7]는 메시지 전송에 의한 병렬 컴퓨팅을 대상으로 노드간 통신에 필요한 네트워크 인터페이스 구현을 목표로 한다. 즉 임계성이 뚜렷한 병렬 응용들을 효과적으로 지원하기 위한 네트워크 프로토콜 구조를 구현한다. 이를 위해서 BIP는 zero memory copy, big message delivery based on pipeline, small message delivery등의 기술들을 구현하였다.

3. 구현범위 및 개발환경

3.1 시스템 환경

CROWN은 Linux 기반의 Pentium 급 PC 워크스테

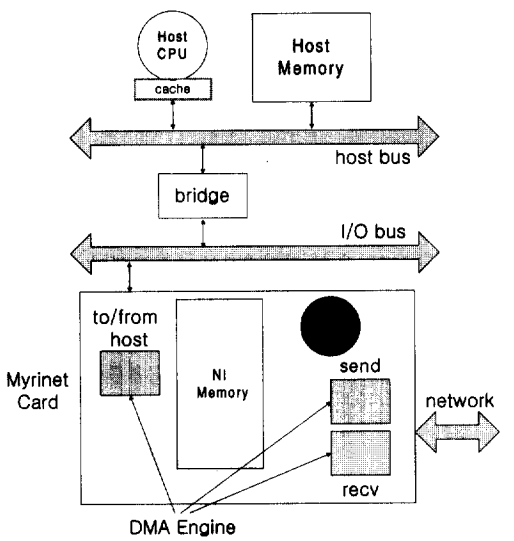
이션을 하드웨어 플랫폼으로 갖는다. 자세한 개발환경은 <표 1>과 같다. 모든 CLCP 모듈의 개발은 GNU-C 컴파일러를 이용했으며, Myrinet 카드의 제어 프로그램은 Myricom에서 제공하는 Lanai용 GNU C 크로스 컴파일러를 사용했다.

〈표 1〉 CROWN 시스템 개발 환경

ITEM	SPEC.
CPU	233MHz Pentium Pro+
Memory	128Mbytes
Disk	Ultra Wide SCSI Disk
Intra-Cluster Network	Myrinet
Server-to-Client Network	Fast Ethernet, Ethernet, or ATM
Operating System	Linux kernel version 2.0.x

3.2 NIC (Network Interface Card) : Myrinet

Myrinet[9]은 양방향 2.56 Gbps를 지원하는 스위치 기반 네트워크이다. Myrinet은 가변 길이의 패킷을 지원하며, 신뢰도가 높은 케이블과 크로스바 스위치를 통해서 웜홀(wormhole) 라우팅을 한다.



(그림 1) 호스트와 Myrinet 구조

(그림 1)은 Myrinet 클러스터 시스템의 한 노드의 구조를 보여준다. Myrinet을 통한 모든 패킷 송수신 작업은 Myrinet의 메모리를 경유해서 일어난다. Myrinet 카드와 호스트는 DMA기능을 통해서 서로의 메모리에 접근이 가능하며, 호스트에서는 DMA 부담을 줄이기 위해서 PIO(Programmed I/O)를 할 수 있다. Myrinet은 프로그래머블 펌웨어(Programmable firmware)를 지원한다. 그러므로, 개발자는 NI(Network Interface)의 자료구조를 개발 전략에 따라 재 설계할 수 있고, 제어 프로그램을 구현할 수 있다.

4. CLCP 요구사항

본 장에서는 CLCP가 갖추어야할 기본적인 특성들과 CROWN이 멀티미디어 서버와 병렬 컴퓨팅 서버로 활용될 때의 통신 요구사항에 대해서 알아본다.

4.1 HSCN의 기본적인 특징

일반적으로 잘 알려진 통신 프로토콜(ex. TCP/IP)들은 다음과 같은 소프트웨어적인 부담 때문에 HSCN(High Speed Clustering Network)에 적용하기 어렵다.

- 커널과 사용자간 모드 변환 오버헤드 : 통신 프로토콜이 커널의 내부에 존재할 경우에는 통신을 위해서 항상 커널을 경유해야 한다. 이 경우 발생하는 커널과 사용자 간의 모드 전환(Mode Switching)은 통신 상의 오버헤드를 발생시킨다.

- 메모리 복사 오버헤드(Memory Copy Overhead) : 어플리케이션에서 네트워크로의 메시지 전송 과정에서 발생하는 복사 오버헤드이다. 일반적인 통신 프로토콜(e.g. TCP/IP, …)에서는 응용의 메시지가 네트워크로 나가기 전에 2~3번의 복사가 일어난다. 이러한 오버헤드는 LAN 장비의 성능을 어플리케이션에게까지 전달하지 못하는 주요 원인이 되고 있다.

- 과중한 프로토콜 오버헤드 : 대부분의 통신 프로토콜은 인터넷을 위한 프로토콜이다. 즉, 커다란 망의 운용 및 관리의 관점에서 만들어진 메커니즘이다. 그러므로, 워크스테이션 클러스터링 시스템에서 요구하는 작고 폐쇄된 망에는 불필요한 기능을 많이 가지고 있고, 이러한 기능들은 많은 지연을 갖게 한다.

클러스터링 네트워크를 위한 고속 통신 프리미티브들은 이러한 오버헤드를 줄이기 위해 각각 다음과 같은 방법을 이용하고 있으며, CLCP에서도 같은 방식을 따라야 한다.

- 커널과 사용자간 모드 변환 오버헤드 : 통신 프로토콜의 모든 기능을 라이브러리화 함으로써 사용자

프로그램과 함께 사용자 공간에서 모든 통신 기능을 수행한다.

- 메모리 복사 오버헤드(Memory Copy Overhead) : 사용자 라이브러리화 된 통신 프리미티브를 이용해서 전송할 자료를 사용자 공간에서 NIC로 직접 처리하면 메모리 카피 수를 1번 이하로 줄일 수 있다.
- 과중한 프로토콜 오버헤드 : 인터넷을 위한 프로토콜에서 LAN 과 워크스테이션 클러스터링 환경에 필요한 프로토콜을 제외한 모든 부분을 제거한다.

4.2 멀티미디어 스트림을 위한 요구사항

멀티미디어 스트림은 연속적인 데이터의 공급이 이루어져야 하는 특성을 갖는다. 그러므로, CROWN 시스템을 멀티미디어 서버(CROWN Multimedia Server : MovieRo)로 활용할 경우, 기존의 대부분의 고속 통신 프리미티브와 같이 폴링 기반의 통신 방법만을 제공하면 성능상 그 한계를 갖는다. 현재 CROWN 시스템을 기반으로 개발된 VoD 서버 소프트웨어인 MovieRo에서는 하나의 클라이언트에 하나의 클라이언트 세션 쓰레드(Client Session Thread)가 할당되기 때문에 각 시스템 노드마다 동시에 다수의 쓰레드가 실행된다. 폴링 방식을 사용할 경우에는 Linux의 특성상, 하나의 시스템에서 많은 쓰레드가 동시에 실행되면 전체적인 시스템 성능이 크게 저하된다[JUN 98].

이러한 문제를 해결하기 위한 방안의 하나로 Ethernet 카드의 인터럽트 기능의 도움을 이용할 수가 있다. 즉 TCP/IP 패킷을 신호 메시지로 이용하여 폴링을 하지 않고도 메시지의 수신여부를 파악할 수 있다[JUN98-1]. 그러나, 이 방법을 사용할 경우 TCP/IP 패킷을 매번 생성해야 한다는 부담을 안게 된다. CLCP는 문제를 해결하기 위해서 인터럽트 기능을 갖도록 설계 및 구현되어야 한다. 인터럽트 기능을 추가하기 위해서는 디바이스 드라이버의 인터럽트 처리 함수, 사용자 프로세스를 깨우기 위한 기능, Firmware의 선택적 인터럽트 기능과 이를 수용하기 위한 새로운 송수신 방법 등이 필요하다.

4.3 병렬 컴퓨팅 환경을 위한 요구사항

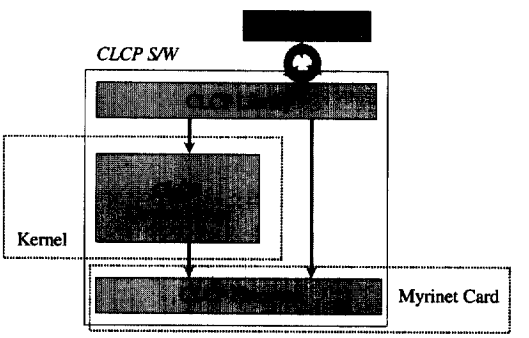
2장의 관련연구에서 거론된 대부분의 통신 프리미티브들은 병렬 컴퓨팅 서버를 목적으로 개발되었다. 병렬 컴퓨팅을 위한 가장 중요한 통신 요구사항은 작은 지연시간(Latency)이다. 메시지 크기에 따른 지연시간의 추이에 따라서 통신 시스템의 성능이 평가된다. 지연시간을 줄이기 위해서 통신 메커니즘에서 취하고 있는 방식은 여러가지가 있다. 우선 송신하는 입장에서는 작은 메시지의 고속 통신을 위해서 FM에서는 PIO (Programmed I/O)방식을 이용하며, 큰 메시지의 송수신을 위해서는 DMA방식을 이용한다. 수신의 입장에서는 인터럽트 방식을 사용할 경우 인터럽트 자체의 부담으로 인해 지연시간이 길어진다. 그러므로, 폴링방식을 이용해서 메시지의 도착을 기다리고, 도착하자마자 컴퓨팅 작업을 시작하는 방식이 적당하다. 높은 성능을 얻기 위해서 폴링 방식을 적용하려면 앞 단원에서 기술한 인터럽트 방식과 달리 각 호스트 마다 하나의 프로세스만이 실행하는 병렬 컴퓨팅 환경을 가져야 한다.

5. CLCP 설계 및 구현

CLCP는 CROWN 시스템의 클러스터 내주 고속 통신을 위해서 구현된 메커니즘이다. 본 장에서는 CLCP의 기본적인 자료구조를 포함한 설계 내용을 설명하고 구현에 관한 자세한 사항을 각 모듈의 관점과 알고리즘 관점에서 기술한다. 그리고, 구현상의 기술을 자세히 알아본다.

5.1 소프트웨어 모델

CLCP는 (그림 2)와 같이 3개의 소프트웨어 모듈로 구성된다. CLCP 라이브러리는 CLCP의 API(Application Programming Interface)를 제공한다. 여기에는 통신 초기화 인터페이스, 인터럽트를 이용한 송수신 인터페이

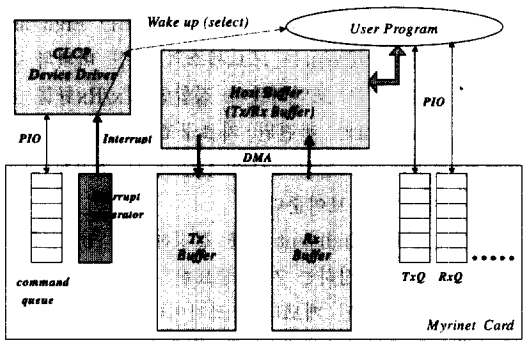


(그림 2) CLCP의 소프트웨어 모델

스, 폴링을 이용한 송수신 인터페이스, 그리고 CLCP SAR(Segmentation And Reassembly) 계층 인터페이스 등이 있다. CLCP 디바이스 드라이버는 Linux 커널에 CLCP 드라이버 모듈을 로드 하는 기능, 디바이스의 초기화 작업 수행 기능, 메모리 매핑 기능 등을 수행하며, 실제 통신이 일어날 때 통신 초기화 작업을 수행한다. 또한 NIC에 필요한 명령을 전달하여 카드를 제어하는 작업도 수행한다. CLCP Firmware는 Myrinet 카드의 메모리에 상주하는 프로그램으로서, 송수신을 위해서 NIC에 생성된 자료구조를 관리하고, 호스트와 혹은 네트워크와의 DMA작업을 수행한다.

5.2 빌딩블럭

(그림 3)은 CLCP에서 하나의 프로세스가 통신을 할 때의 빌딩블록을 보여준다. 모든 송수신 프로세스는 통신을 하기 위해서 하나 이상의 EndPoint를 갖는다. EndPoint는 송수신을 위한 버퍼와 큐를 갖는 자료구조이며, 뒤에서 자세히 설명하기로 한다. (그림 3)에서 Tx Buffer, Rx Buffer는 NIC의 메모리에 각각 하나씩만 존재하는 고유의 통신 버퍼이다. 한 호스트 내의 모든 패킷들은 이 버퍼를 경유해서 송수신을 한다. 즉 CLCP는 이 버퍼를 통해서 멀티플렉싱/디멀티플렉싱을 하며 동시에 여러 개의 통신채널을 제공한다. 이 버퍼에 있는 패킷들은 (그림 1)의 DMA 엔진에 의해서 네트워크나 호스트로 DMA 된다. NIC에 있는 Tx/Rx Queue와 호스트 메모리에 있는 Host Buffer는 각각의 통신 프로세스에 하나씩 존재한다. NIC는 도착하는 패킷을 보고 호스트로 하드웨어 인터럽트를 발생시킬 수 있다. 인터럽트 시그널이 통신 프로세스에게 까지 전달되기 위해서 CLCP는 select 시스템 호출을 사용한다. NIC에 있는 Command Queue는 통신 초기화 및 해제 작업을 할 때, 자료구조의 생성 및 소멸을 호스트로부터 지시받기 위해 사용되는 큐이다.



(그림 3) CLCP 빌딩 블록

5.3 각 구성 모듈의 기능

5.3.1 CLCP 라이브러리

CLCP 라이브러리에서 지원하는 API는 <표 2>와 같다. CLCP의 MTU는 4K Bytes이다. 이것은 하나의 패킷을 위한 통신 버퍼의 크기를 4K Bytes로 제한한다. 그러므로, 통신 프로세스에서는 4K 이상의 패킷은 버퍼에 맞게 세스멘테이션 작업을 해주어야 한다. CLCP에서는 CLCP의 버퍼 크기에 의존하지않고 가변적인 패킷을 메시지라하고, CLCP의 버퍼에 맞게 세그멘테이션된 패킷을 프레임이라고 한다. <표 2>의 API에서 message 혹은 msg가 포함된 이름을 갖는 API는 통신 프로세스에서 세그멘테이션을 하지 않는 경우 사용하는 API로서 그 함수 안에서 SAR(Segmentation And Reassembly) 작업을 수행해 준다. 이 API들은 한 번의 메모리 복사 부담을 갖는다.

〈표 2〉 CLCP API

API		기 능
통신 초기화	clcp_open	디바이스 파일 열기
	clcp _alloc_endpt	하나의 EndPoint 할당
	clcp _alloc_chan	하나의 채널 할당
통신 해제	clcp_close	디바이스 파일 닫기
	clcp_release_endpt	할당했던 EndPoint를 Free
	clcp_release_chan	할당했던 채널을 Free
인터럽트 송수신	clcp_send_frame	프레임(CLCP MTU)단위의 송신
	clcp_recv_frame	프레임 단위의 수신
	clcp_send_message	가변길이 메시지 송신 (CLCP SAR API)
	clcp_recv_message	가변길이 메시지 수신 (CLCP SAR API)
	clcp_release_frames	통신버퍼의 상태를 수신 가능 상태로 전환
폴링 송수신	clcp_poll_tx_msg	폴링 기반의 가변 메시지 송신
	clcp_poll_rx_msg	폴링 기반의 가변 메시지 수신

5.3.2 CLCP 디바이스 드라이버

CLCP 디바이스 드라이버는 <표 3>과 같이 5개의 UNIX 표준 디바이스 인터페이스(file_operations)만을 구현했다. select 함수는 CLCP의 인터럽트 기능을 위해 구현되었다. 인터럽트를 이용한 통신 프로세스는 자료의 수신을 위해서 select 시스템 호출을 한 후 시

스템에서 블록된다. 메시지가 수신되면 NIC는 인터럽트를 통해서 수신 프로세스의 정보를 커널에게 전달한다. 커널은 수신 프로세스를 대기 상태로 전환 시켜서 스케줄링에 의해 일어날 수 있게 한다. 인터럽트에 의한 수신 과정과 자료구조는 뒤에서 자세히 알아보기로 한다.

〈표 3〉 CLCP 디바이스 드라이버의 UNIX 표준 인터페이스

Interface functions	Implementation
lseek	X
read	X
write	X
readdir	X
select	O
ioctl	O
mmap	O
open	O
close	O
fsync	X
fasync	X
check_media_type	X
revalidate	X

mmap 함수는 통신 프로세스에서 자유로이 커널과 NIC 영역에 존재하는 버퍼를 접근할 수 있도록 사용자 가상 주소체제로 변환시켜주는 역할을 한다. 메모리 매핑에 관한 절차와 방법 역시 뒤에서 자세히 다룬다.

ioctl 함수는 CLCP가 가지고 있는 기타 기능들의 인터페이스 역할을 한다. ioctl 함수에 의해서 CLCP 디바이스 드라이버에 연결되는 함수들은 통신 초기화와 관련된 함수들로 기능은 <표 4>와 같다.

상기한 4개의 함수들은 통신 초기화/해제 과정에서 자료구조를 생성/소멸 하는 일을 처리한다. 그러므로, 이 함수들은 각각의 통신을 위해서 Firmware에 필요한 자료구조의 생성/소멸 역시 명령한다.

〈표 4〉 ioctl 함수에 연결된 디바이스 함수들

함 수	기 능
Clcp_create_endpt	커널 메모리와 NIC에 EndPoint 생성
Clcp_destroy_endpt	EndPoint 소멸
Clcp_activate_chan	채널 할당
Clcp_deactivate_chan	채널 소멸

이 밖에 CLCP 디바이스 드라이버에서는, NIC에서 호스트로 인터럽트가 발생했을 때 이를 처리하는 작업(clcp_interrupt 함수)을 하며, 디바이스 드라이버를 커널로 로딩한 후 디바이스의 초기화 작업을 수행한다(init_module 함수).

5.3.3 CLCP Firmware

CLCP F/W(Firmware)는 Myrinet 제어 프로그램으로서, 호스트에서 실행되는 로딩 프로그램에 의해서 Myrinet 메모리로 로딩된다. F/W는 NIC의 메모리에 있는 자료구조를 활성(Enable)/비활성(Disable) 상태로 만들고, 통신 프로세스의 큐를 폴링한다. NIC 메모리에는 실행중인 각각의 통신 프로세스들이 가지고 있는 Tx/Rx 큐들과 CLCP 디바이스 드라이버와의 명령 및 신호 교환을 위한 명령큐(Command Queue)가 있다. F/W는 현재 활성(Enable)상태인 Tx큐들과 명령큐를 순차적으로 폴링한다. 또한, F/W는 패킷의 수신을 지속적으로 모니터링하고 있어야 한다. 패킷이 도착하면 하드웨어적으로 NIC의 레지스터인 ISR(Interrupt Status Register)의 BYTE_RDY_BIT이 세팅된다. 그러므로, F/W에서는 BYTE_RDY_BIT의 세팅여부를 지속적으로 체크한다.

F/W에서는 CLCP 디바이스 드라이버와 공유하는 자료구조를 갖는다. 공유되는 자료구조에는 명령큐의 베이스 주소와 포인터, F/W의 디버깅 시에 활용할 로그 버퍼, 라우팅 정보, 인터럽트시 전달될 정보와 동기화 변수 등이 포함된다.

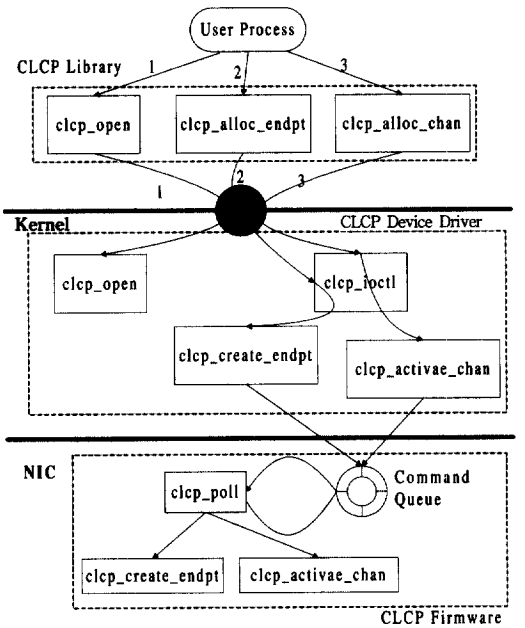
5.4 CLCP 알고리즘

본 장에서는 상기한 모듈의 함수를 기반으로 통신 과정을 크게 세단계로 나누어서 설명한다.

5.4.1 통신 초기화

통신 프로세스는 통신을 위해서 자료구조를 생성해야 한다. 통신을 위해서 필요한 자료구조는 EndPoint로 대표된다. EndPoint는 프로세스의 통신 버퍼와 그 버퍼를 포인트하는 포인터들의 집합인 Tx/Rx큐를 포함한다. EndPoint와 관련된 자료구조는 뒤에서 다시 자세히 설명하기로 한다. (그림 4)는 통신 초기화 과정의 함수 흐름도를 보여준다.

CLCP는 최대 EndPoint 수를 결정해놓고 있다. 최대 EndPoint 수를 결정하는 가장 중요한 요소는 호스트 메모리에 Pin-down된 메모리의 크기이다. Bigphysarea-patch루틴은 Pin-down메모리를 위해서 Matt Welsh에 의해서 고안된 Linux 커널 패치루틴이다. Bigphysarea



(그림 4) 통신 초기화과정의 함수 흐름도

에 대한 설명은 메모리 매핑에 관한 단원에서 자세히 하기로 한다. 개발자는 Linux의 부팅 이미지를 편집해서 Pin-down메모리의 크기를 조정할 수 있다.

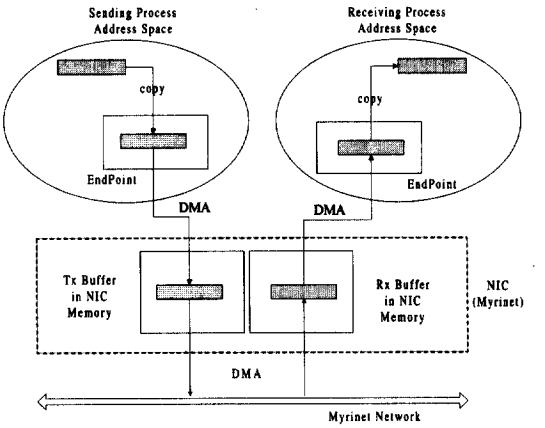
각각의 EndPoint는 하나의 디바이스 파일로 표현된다. 예를 들어, 최대 EndPoint수를 16이라고 했을 때, /dev 디렉토리에 clcp0 부터 clcp15까지의 파일을 생성할 수 있다. 하나의 EndPoint를 사용하기 위해서는 하나의 디바이스 파일을 열어야 한다. clcp_open은 디바이스 파일을 여는 작업을 한다.

clcp_alloc_endpt API는 ioctl 인터페이스를 통해 CLCP 디바이스 드라이버의 clcp_create_endpt 함수에 연결된다. 이 함수에서는 bigphysarea 영역에서 메모리를 할당받고, CLCP F/W에 EndPoint 생성을 명령한다. NIC와 디바이스 드라이버에서 할당된 메모리들은 사용자가 사용할 수 있도록 사용자 주소공간에 매핑(Mapping)된다. 메모리 매핑(Memory Mapping)에 관한 내용은 구현 기술 부분에서 자세히 다룬다.

clcp_alloc_chan API는 통신 ID역할을 하는 채널을 할당받는다. 채널에 대한 설명은 EndPoint에 대한 부분에서 자세히 설명한다.

5.4.2 송수신 과정

CLCP를 경유하는 패킷은 커널을 우회하는 통신 경로를 갖는다. (그림 5)는 호스트 내에서 패킷의 흐름을 보여준다.



(그림 5) CLCP 패킷 흐름도

<표 2>에서와 같이 CLCP에는 폴링과 인터럽트를 이용한 두종류의 송수신 인터페이스가 존재한다. 패킷의 흐름은 두가지 인터페이스 모두 (그림 5)와 같다.

송신을 위해서 통신 프로세스는 먼저 송신하고자 하는 패킷을 자신의 EndPoint의 송신버퍼에 복사를 해야 한다. 복사된 패킷의 정보는 EndPoint에 있는 전송큐(Tx Queue)에 기록되어야 한다. 복사 및 정보 기록 작업은 송신 API에서 수행한다. EndPoint에 패킷이 복사되면 사용자 수준의 송신은 마무리된다. 송신 API에서 제어가 리턴되면 통신 프로세스는 다른 작업을 수행한다. 패킷을 실제 네트워크로 전송하는 작업은 F/W에 의해서 수행된다. 모든 통신 프로세스의 전송 및 수신 큐는 물리적으로 NIC의 메모리에 존재하므로, F/W는 이 큐들에 쉽게 접근할 수 있다. F/W는 NIC에 존재하는 명령큐를 포함한 모드 큐를 폴링하는 작업을 되풀이한다. 이때, 송신할 패킷이 있음을 알게되고, 큐에 있는 패킷의 정보(DMA 주소, 크기, 채널번호 등)를 보고 DMA를 통해 NIC의 통신 버퍼로 복사한 후, 다시 네트워크로 DMA를 통해 전송을 한다.

수신작업을 위해서 수신 프로세스는 자신의 EndPoint에 있는 수신큐(Rx Queue)에 도착한 패킷이 복사될 EndPoint의 수신 가능 버퍼 공간 정보를 미리 기록해둔다. 인터럽트를 이용한 수신을 원할 경우에 수신 프로세스는 select 시스템 호출을 해야한다. 인터럽트를 통한 수신 과정은 다음 단원에서 자세히 알아본

다. 수신을 위한 준비가 된 상태에서 패킷이 도착하면 F/W는 도착된 패킷의 헤더 정보를 보고 패킷을 수신할 목적 프로세스의 EndPoint를 찾아낸다. 그리고, EndPoint에 있는 수신큐를 보고 패킷이 DMA될 물리 주소를 알아낸 후 EndPoint의 수신버퍼로 DMA를 한다. 수신 프로세스는 자신의 수신큐의 OWNER_FLAG를 보고 패킷이 도착했음을 알고, 도착된 패킷을 자신의 다른 주소 공간으로 복사한 후 사용된 수신버퍼 공간을 다시 수신가능 버퍼 공간으로 환원한다.

5.4.3 통신 해제

통신 해제 작업은 통신에 사용되었던 EndPoint를 시스템에서 다른 통신을 위해서 사용할 수 있도록 반납하는 작업이다. 이 과정은 통신 초기화 작업을 역으로 수행한다.

5.5 구현 기술 및 자료 구조

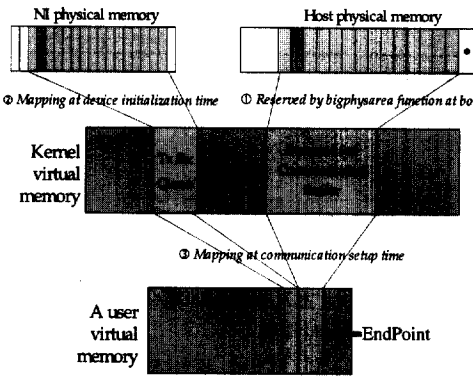
CLCP를 구현하는데는 몇가지 중요한 구현상의 기술이 필요하다. 본 단원에서는 앞에서 기술한 각 모듈의 기능과 통신을 위한 절차들에서 부분적으로 설명된 구현 기술과 자료구조들을 설명한다.

5.5.1 메모리 매핑

CLCP에서 통신을 하기 위해서 가장 중요한 자료구조는 EndPoint이다. EndPoint는 통신을 위한 버퍼와 큐들로 구성되어있다. EndPoint는 사용자 주소 공간에서 일련된 주소로 접근이 가능하다. 그러나, EndPoint는 물리적으로 Pin-down된 커널영역의 메모리와 NIC에 있는 메모리로 구성된다. 이와 같이 물리적으로, 그리고 운영체제의 기본적인 전략상으로 접근이 금지되어있는 영역을 사용자가 자유로이 접근하기 위해서는 (그림 6)과 같은 메모리 매핑(Memory Mapping)과정이 필요하다.

I/O 주변장치와 DMA를 이용한 입출력을 하기 위해서는 호스트 메모리의 일정부분이 Pin-down되어져야 한다. 네트워크와 같이 대용량의 입출력이 필요한 경우 Pin-Down되야 할 메모리의 크기는 더욱 커진다. CLCP에서는 이를 해결하기 위해서 Bigphysarea라는 Pin-down 영역을 갖는 Linux 버전을 사용한다. Bigphysarea는 시스템이 부팅할 때 커널 가상 메모리에 큰 배열을 잡음으로써 시스템이 부트업(Boot-Up)된 후에 물리적으로도 스왑 아웃(Swap-out)되지 않게 되는 메모리 영역이다.

CLCP의 통신 프로세스는 NIC의 메모리에도 접근을 할 수 있어야 한다. 이를 위해서는 우선 NIC 메모리에 커널 가상 주소를 부여해야 한다. Linux에서는 이를 위해서 vremap 커널 함수를 제공한다. CLCP의 디바이스 드라이버가 커널에 로드되는 과정에서 vremap 함수가 호출되고 (그림 6)의 ②번 과정이 진행된다. 이 과정이 끝나면 EndPoint를 위한 메모리 공간은 커널 주소 공간으로 접근이 가능하게 된다.



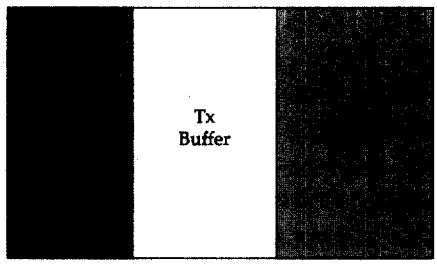
(그림 6) CLCP의 메모리 매핑 과정

통신 프로세스는 EndPoint를 할당받은 후 사용자 공간의 주소로 EndPoint에 접근을 하기 위해서 mmap 함수를 사용한다. mmap시스템 호출과 매치되는 CLCP 디바이스 드라이버 함수는 clcp_mmap이다. clcp_mmap은 두번의 메모리 매핑작업을 한다(그림 6의 ③). 하나는 bigphysarea로부터 할당받은 버퍼 영역이고, 다른 하나는 NIC 메모리에 있는 큐를 위한 공간이다. 이때 사용되는 커널 함수는 remap_page_range 함수이다. 두번의 매핑작업을 끝내면 사용자 가상 주소공간에는 EndPoint가 연속적인 주소를 갖는 공간으로 보인다.

5.5.2 EndPoint

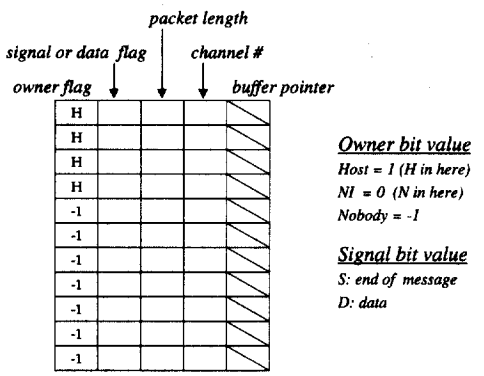
앞에서 계속 언급했듯이 EndPoint는 CLCP 통신의 핵심적인 자료구조이다. EndPoint를 관리하는 자료구조는 통신프로세스와 CLCP 디바이스 드라이버, 그리고 F/W가 모두 가지고 있다. (그림 7)은 사용자 관점에서 본 EndPoint이다.

(그림 7)과 같이 하나의 EndPoint는 송신/수신 큐와 송신/수신 버퍼로 구성된다. 네개의 컴포넌트는 연속된



(그림 7) 사용자 관점의 EndPoint 구조

사용자 주소공간에 존재한다. 송수신 큐의 구조는 (그림 8)과 같다.



(그림 8) 송수신 큐의 구조

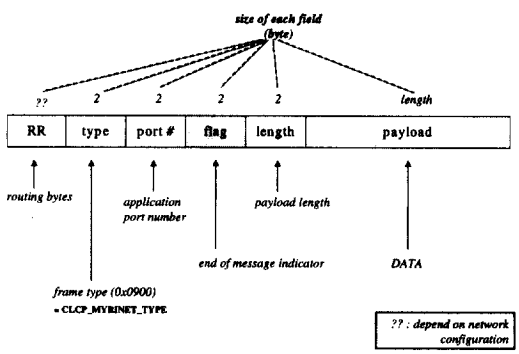
송수신 큐는 송수신을 위해서 사용자 프로세스와 F/W가 정보를 교환하는 창구 역할을 한다. 이를 위해, 각각의 큐 원소는 세가지 소유자 상태를 가진다. 소유자 상태 필드(owner flag)의 값은 이 원소가 가지고 있는 값이 누구에게 유효한가를 의미한다. 사용자 프로세스에서 전송하고자 하는 메시지는 CLCP를 이용해서 전송되기 위해서 SAR계층을 경유한다는 것은 앞에서 설명한 바와 같다.

그러므로, 사용자가 전송하고자 하는 메시지는 하나 이상의 CLCP 프레임으로 구성된다. 신호필드(signal flag)는 현재 프레임이 메시지의 끝인지 아닌지를 나타낸다. 이 필드는 특히 인터럽트를 이용한 송수신을 위해서 고안되었다. 채널번호 필드는 버퍼 포인터 필드에서 가리키는 프레임이 전송될/전송한 노드와 상대의 포트 번호를 찾는 키로 사용된다. 버퍼 포인터 필드는 프레임이 저장된/저장될 EndPoint 내의 위치를 알려준다. 단원의 서두에 언급한 바와 같이 EndPoint는 CLCP의 세가지 소프트웨어 모듈 모두에서 관리하는 자료구조이다. 그러므로, 각각의 모듈에서 접근하는 주소 공간(Address Space)가 모두 다르다. 그렇기 때문에, 송수신 큐에 사용되는 모든 위치 정보는 오프셋으로 표기된다.

CLCP 라이브러리에서 EndPoint로 접근하기 위해서 사용하는 자료구조는 user_endpt이다. user_endpt에는 사용자 가상 주소로 EndPoint를 접근할 수 있도록, 송수신 큐의 포인터(first, next, last), 버퍼의 베이스 주소, 버퍼의 크기와 개수등의 정보를 갖는다. CLCP 디바이스 드라이버에서 갖는 자료구조는 인터럽트를 위한 Wait 큐, 채널관리를 위한 채널 배열, 초기화와 해제 과정에 필요한 호스트 메모리에 있는 EndPoint의 물리주소와 크기등을 갖는다. CLCP F/W에서는 사용자 프로세스와 같이 큐에 접근하기 위한 큐에 대한 포인터들과 호스트로/부터 의 DMA를 위한 EndPoint 버퍼의 베이스 주소를 갖는다. 또한, 채널관리를 위한 채널 배열을 갖는다.

5.5.3 패킷 구조

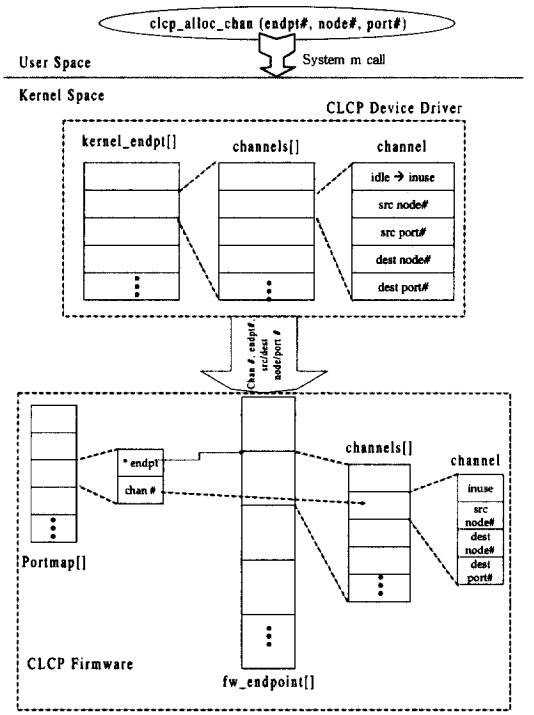
(그림 9)는 CLCP 패킷의 헤더 구조를 보여주고 있다. CLCP 헤더의 제일 앞부분은 목적지 노드로 전송되기 위한 라우팅 정보가 들어간다. CLCP는 Myrinet을 이용하고, Myrinet은 Wormhole 라우팅을 하는 스위치 기반 네트워크이다. CLCP는 F/W를 NIC 메모리에 로딩할 때 망의 컨피그 정보를 입력받는다. 이 입력 정보를 기반으로 라우팅 테이블을 생성한다. 이 방식은 FM(Fast Messages)에서 사용한 방식을 이용했다.



(그림 9) CLCP 패킷 헤더 구조

port # 필드는 목적 프로세스의 포트번호를 갖는다.

포트번호와 채널과의 관계는 다음 절에서 자세히 다룬다. flag 필드는 이 프레임이 마지막 메시지인지 아닌지를 나타내며, 이 정보에 따라 인터럽트의 발생 여부가 결정된다.



(그림 10) 채널 설정 과정

5.5.4 채널

CLCP는 통신을 위해서 채널 개념을 사용한다. 일반적으로 통신 프로세스가 통신을 하기 위해서는 상대의 네트워크 주소와 프로세스 주소를 알아야한다. 사용자는 CLCP에게 이를 표현하기 위해서 노드번호와 포트번호를 사용한다. CLCP에서는 포트번호를 원활히 관리하기 위해서 하나의 포트번호당 하나의 채널을 매핑한다. 이 과정은 clcp_alloc_chan 함수에 의해서 진행된다. 채널을 할당하는 작업은 여러 개의 자료구조에 의해서 일어난다. (그림 10)은 이 과정의 예를 보여준다. (그림 10)에서 보듯이 하나의 EndPoint에는 여러 개의 채널이 존재한다. 즉, CLCP에서는 하나의 EndPoint를 이용해서 다중통신을 할 수 있다. 사용자 프로세스는 CLCP API의 clcp_alloc_chan 함수를 이용해서 채널 할당을 요구한다. 이때, 채널을 요구하는 EndPont 번호와 통신할 목적지 및 자신의 노드번호/포트번호를 넘겨준다. 디바이스 드라이버에서는 요청된 EndPoint 내에서 사용되지 않는 채널을 찾아서 구조체를 세팅한 후, F/W에게 할당된 채널 번호와 EndPoint 번호등을 넘겨주고 해당작업을 명령한다. F/W는 자신이 관리하는 EndPoint 구조체의 매치되는 채널 구조체를 세팅하고, 자신의 포트번호를 인덱스로 하는 Portmap 구조체에 EndPoint정보와 채널번호를 세팅한다.

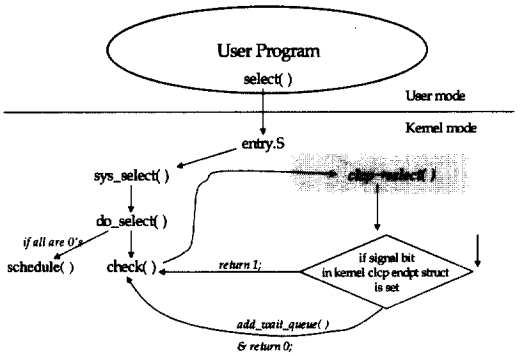
이와 같이 할당된 채널은 송수신 과정에서 상대 프로세스(Peer Process)를 표현한다. 송신할 때 송신큐에는 패킷의 목적지를 나타내는 정보로 채널번호를 가지고 있다. F/W에서는 fw_endpoint[endpt#].channels[chan#].dest_port#를 패킷의 헤더에 복사해서 패킷을 전송한다. 수신시에는 도착된 패킷의 헤더에 있는 포트번호를 보고, 채널번호와 EndPoint구조체의 주소를 알아낸다(Portmap[port#].chan,# Portmap[port#].endpt).

5.5.5 인터럽트

기본적으로 고속 통신 프리미티브에서는 인터럽트 방식을 사용하지 않고 있다. 인터럽트 자체의 부담이 지연시간에 큰 부담으로 작용하기 때문이다. 실제로 인터럽트 자체의 소요시간만도 10μs 이상인 것으로 알려져 있다. 그러나, 요구사항에서 기술했듯이 멀티프로세싱 환경에서 여러 개의 프로그램이 동시에 실행되고 있을 경우에, 폴링방식을 사용할 경우 통신 성능은 크게 저하된다. 특히, CROWN 시스템의 VOD 서버같이 여러 개의 프로세스가 동시에 실행되는 멀티미디어 스트림 서버들을 위해서는 인터럽트 방식이 꼭 필요하다.

CLCP에서 실제로 인터럽트는 F/W에 의해서 발생되지만, 인터럽트의 발생여부는 패킷을 송신하는 프로세스에 의해서 결정된다. 앞에서 언급한대로 CLCP F/W에서는 패킷의 헤더의 flag 필드 값에 의해서 인터럽트 발생 여부를 결정한다. 이 필드의 값을 세팅하는 권한은 송신측의 사용자 프로세스가 갖고 있다.

인터럽트를 이용하는 기술적 이유는 패킷의 수신시에, 패킷의 도착한 후 폴링을 하므로써 폴링 부담을 최소화하기 위해서이다. 그러므로, 통신을 하는 사용자 프로세스들은 패킷이 도착할 때까지 프로세스 큐에서 블록상태(Wait Status)로 있게 된다. 패킷이 도착하면 해당 프로세스는 커널에 의해 깨워서 대기상태(Ready Status)로 전환된다.



(그림 11) CLCP에서 select 시스템 호출시 함수 흐름

CLCP에서는 이 과정을 select 시스템 호출을 이용하였다. (그림 11)은 CLCP에서 select 시스템 호출을 이용할 경우의 함수 흐름도이다.

NIC에서 하드웨어 인터럽트가 발생하면 clcp_interrupt 루틴에서는 kernel_endpt 구조체에 있는 select 필드를 세팅하게 되고, 커널에서 다음에 clcp_select 함수를 호출할 경우 clcp_select 함수는 패킷이 도착했음을 알고 1을 리턴하게 된다. 이때 커널은 해당 프로세스를 대기상태로 전환한다.

6. 성능 평가

CLCP는 폴링과 인터럽트 방식이 성능에 영향을 주는 사용자 수준 하위 계층 통신 프리미티브이다. 본 장에서는 CLCP에서 제공하는 인터럽트 방식과 폴링 방식의 성능을 비교 분석한다.

실험은 두 대의 호스트 사이에서 진행되었다. 각 호스트에는 CLCP 통신을 위한 프로세스가 기본적으로 존재하며 실험을 위한 실행 패턴(3ms Running/50ms Block)이 조절된 널(Null) 프로세스들이 존재할 수 있다. 이와 같은 실행 패턴은 CROWN 멀티미디어 서버의 클라이언트 세션 쓰레드의 실행 패턴을 에뮬레이션한 것이다. 모든 실험 결과는 메시지의 송수신을 1000번 반복한 후 측정한 시간을 기반으로 계산되었다.

실험 결과는 (그림 12)와 같다. 즉 8K Bytes(CLCP MTU)이하의 메시지 송수신이 잦은 병렬 컴퓨팅 서버의 경우에는 폴링 방식을 택하는 것이 성능이 우수하다(그림 12 (a),(b)). 메시지 길이가 긴 경우에도 시스템에 다른 실행 프로세스가 없는 경우에는 폴링방식의 성능이 좋게 나옴을 알 수 있다(그림 12 (c),(d)). 그러나, 프로세스가 증가하면서 폴링과 인터럽트의 성능 격차가 점점 줄어들며, 시스템에 프로세스의 수가 90개에 이를 경우 인터럽트 방식의 성능이 폴링 방식의 성능을 현저히 능가함을 볼 수 있다(그림 12 (e),(f)).

7. 결론 및 향후 연구방향

본 논문에서는 CROWN 시스템의 내부 통신 프리미티브인 CLCP의 구현 내용에 대해서 설명하였다. CLCP는 하위 계층의 통신 성능을 상위 응용 계층까지 전달하는 역할을 하며, 병렬 컴퓨팅을 위한 클러스터링 내부 통신 요구사항과 멀티미디어 스트림을 위한 요구사항을 지원한다.

호스트에 다른 프로세스가 없는 경우 폴링 대역폭은 최고 450Mbps 수준까지 달하지만, 프로세스가 많아질수록 성능이 크게 저하됨을 알 수 있었다. 인터럽트 방식의 경우, 다른 프로세스가 없을 때 인터럽트 자체 오버헤드로 인해 성능이 폴링에 비해서 떨어지지만, 프로세스가 많아짐에 따라 발생하는 성능 저하 곡선이 폴링보다 완만함을 알 수 있었다.

현재 개발된 CLCP는 기존의 고속 통신 프리미티브들, AM, FM, PM, VMMC … 등에 비해서 성능이 대체적으로 떨어지는 편이다. 그러나, 폴링 방식의 통신을 하고 있는 기존의 메커니즘에 반해 인터럽트를 지원하므로서 보다 융통성있게 클러스터링 시스템의 응용 레벨을 지원할 수 있다.

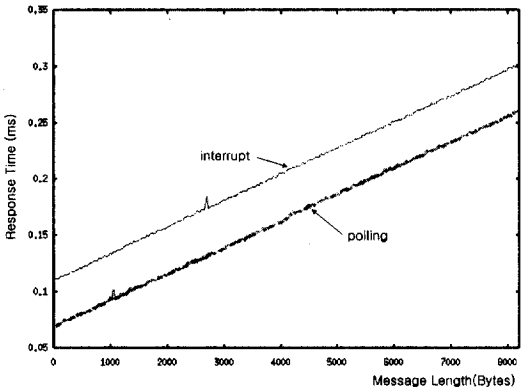
향후 CLCP의 성능 개선을 위해서 인터럽트 방식을 개선하고 NIC 수준의 이중 버퍼 관리 방법이 적용될 예정이다. 또한, 기능적인 측면에 있어서도 클러스터 내부의 멀티캐스팅, QoS, 흐름제어 기법 등에 대한 연구도 진행될 예정이다.

참 고 문 헌

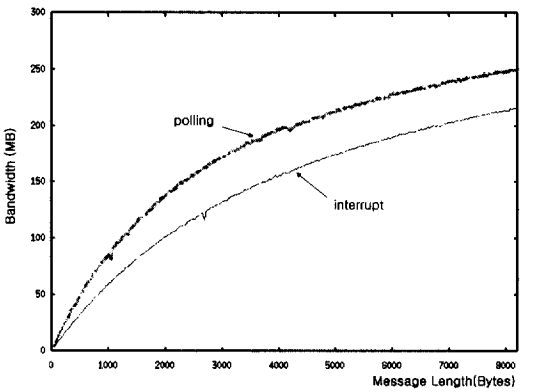
[1] J. H. Park, K. D. Moon, T. Kim, G. H. Cho, "Performance Characteristics of Polling and Interrupt for a Clustered Video Server," Proceedings of ICMTM'98, pp.470-478, 1998.

[2] 박준희, 문경덕, 김태근, 조기환, "클러스터 기반 멀티미디어 서버를 위한 고속 통신 프리미티브 설계", 한국통신학회 추계 학술대회 학술지, 1998.

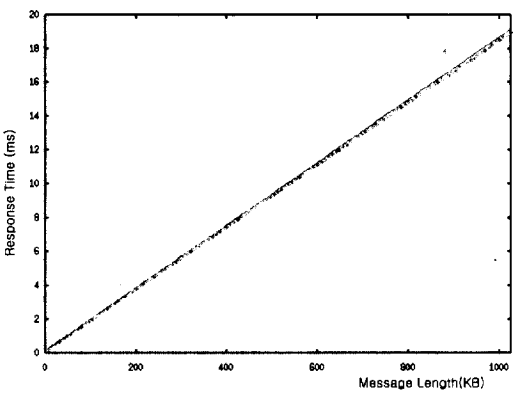
[3] 박준희, 문경덕, 김태근, 박창순, "워크스테이션 클러



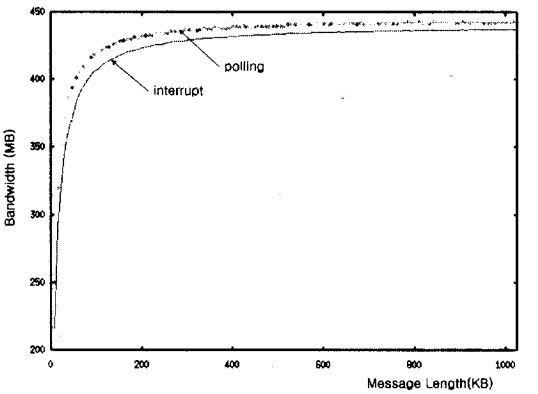
(a) Response time of short packets without null process



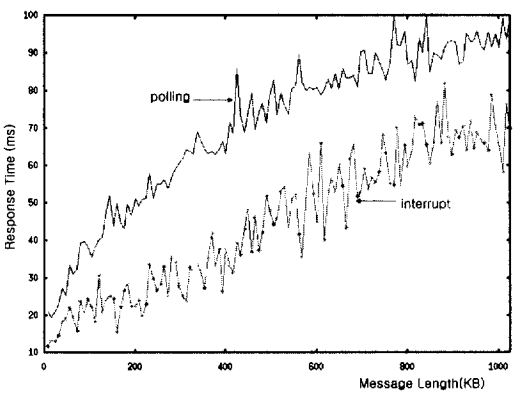
(b) Bandwidth of short packets without null process



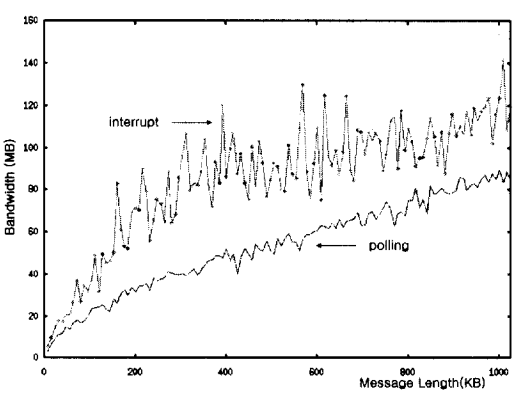
(c) Response time of long packets without null process



(d) Bandwidth of long packets without null process



(e) Response time of long packets with 90 null processes



(f) Bandwidth of long packets with 90 null processes

(그림 12)

스터링 시스템의 초 고속 통신 프리미티브 설계", 한국 정보과학회 추계 학술대회 학술지, 1997.

[4] B. N. Chun, A. M. Mainwaring and D. E. Culler, "Virtual Network Transport Protocols for Myrinet," IEEE Micro Special Issue, Jan/Feb, 1998. http : // now.cs.berkeley.edu/AM/active_messages.html

[5] L. A. Giannini and A. A. Chien, "A Software Architecture for Global Address Space Communication on Clusters : Put/Get on Fast Messages," In Proc. Of HPDC98, 1998. http : //www-csag.cs.uiuc.edu/projects/comm/fm.html

[6] R. Bhoedjang, T. Ruhl and H. E. Bal, "Design Issues for User-Level Network Interface Protocols on Myrinet," IEEE Computer, Nov. 1998. http : //www.cs.vu.nl/~bal/das.html

[7] L. Prylli and B. Tourancheau, "BIP : A New Protocol Designed for High Performance Networking on Myrinet," In Workshop PC-NOW, IPPS/SPDP98, 1998. http ://www-bip.univ-lyon1.fr/overview.html

[8] Basu, A., Buch, V., Vogels, W. and Eicken, T., "U-Net : A User-Level Network Interface for Parallel and Distributed Computing," *15th ACM Symposium on Operating Systems Principles*, Copper Mountain, CO, USA, pp.40-53, 1995.

[9] Boden, N., Cohen, D., Felderman R. and Kulawik, A., "Myrinet : A Gigabit-per-second Local Area Network," *IEEE-Micro*, 15(1), pp.29-36, 1995.

[10] Culler, D., Arpaci-Dusseau, A., Arpaci-Dusseau, R., Chun, B., Lumetta, S., Mainwaring, A., Martin, R., Yoshikawa and C., Wong, F., "Parallel Computing on the Berkeley NOW," *9th Joint Symposium on Parallel Processing*, Kobe, Japan, 1997.

[11] Damianakis, S., Chen, Y. and Felten, E., "Reducing Waiting Costs in User-Level Communication," *11th International Parallel Processing Symposium*, 1997.

[12] Felten, E., Alpert, R., Bilas, A., Blumrich, M., Clark, D., Damiankis, S., Dubnicki, C., Iftode, L., and Li, K., "Early Experience with Message Passing on the SHRIMP Multicomputer," *23rd Annual International Symposium on Computer Architecture, 1996.*

[13] Maquelin, O., Gao, G., Hum, H., Theobald K. and Yia, X., "Polling Watchdog : Combining Polling and Interrupts for Efficient Message Handling," *23th International Symposium on Computer Architecture*, pp.179-188, 1996.

[14] Martin, R., Vahdat, A., Culler, D. and Anderson, T., "Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture," *24th Annual International Symposium on Computer Architecture*, Denver, CO, USA, 1997.

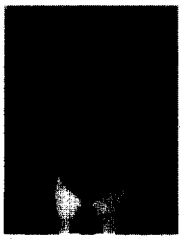
[15] Park, J., Moon, K., Kim, T., and Park, C., "A User-Level Fast Message Receiving Technique by Maintaining Common Available Memory Pages Information in Network Interface," *3rd Asia-Pacific Conference on Communications*, Sydney, Australia, 1997.

[16] Pakin, S., Karamcheti V. and Chien, A., "Fast Messages (FM) : Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors," *IEEE Concurrency*, 5 (2), pp.60-73,1997.

[17] Tezuka, H., Hori, A., Ishikawa, Y. and Sato, M., "PM : An Operating System Coordinated High Performance Communication Library," *Vol.1225 of Lecture Notes in Computer Science*, pp.708-717, 1997.

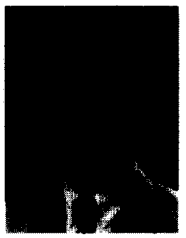
[18] Thomas E. Anderson, David E. Culler, David A. Patterson, "A Case for Networks of Workstations : NOW," IEEE Micro, 1995.

[19] Barak A. and La'adan O., "The MOSIX Multicomputer Operating System for High Performance Cluster Computing," Journal of Future Generation Computer Systems, Vol.13, No.4-5, pp.361-372, March 1998.



박 준 회

e-mail : juni@etri.re.kr
1995년 충남대학교 자연과학대학 컴퓨터과학과(학사)
1997년 충남대학교 자연과학대학 컴퓨터과학과(석사)
1997년~1998년 시스템공학연구소 연구원
1998년~현재 한국전자통신연구원 연구원
관심분야 : 클러스터링 시스템, 홈 네트워크, Mobile IP, 초고속 LAN



문 경 덕

e-mail : kdmoon@etri.re.kr
1990년 한양대학교 전자계산학과 (학사)
1992년 한양대학교 전자계산학과 (석사)
1992년~1997년 시스템공학연구소 연구원
1997년~1998년 시스템공학연구소 선임연구원
1998년~현재 한국전자통신연구원 선임연구원
관심분야 : 클러스터 시스템, 홈 네트워크, 내장형 시스템, 병렬 컴퓨팅

김 태 근

e-mail : tkim@dtvro.com
1987년 한양대학교 수학 학사
1990년 뉴욕주립대학교 전산수학 석사
1993년 뉴욕주립대학교 전산수학 박사
1992년~1998년 시스템공학연구소 분산컴퓨팅연구팀장
1998년~1999년 한국전자통신연구원 실시간컴퓨팅연구부 분산컴퓨팅연구팀장
1999년~현재 (주)DTVRO 대표이사
관심분야 : 병렬처리, 클러스터링 시스템, 정보가전

조 기 환

e-mail : ghcho@cs.chonbuk.ac.kr
1985년 전남대학교 자연과학대학 계산통계학과(학사)
1987년 서울대학교 대학원 계산통계학과(석사)
1995년 Newcastle University, Dept. of Computing Science(Ph.D)
1987년~1997년 한국전자통신연구원 선임연구원
1997년~1998년 목포대학교 전임강사
1999년~현재 전북대학교 조교수
관심분야 : Mobile Computing, 컴퓨터 네트워크, 운영체제