내장형 리눅스를 이용한 라우터의 설계 및 구현

주 민 규[†]·최 경 희[††]·김 종 수[†††]·문 종 욱[†††]·정 기 현[††††]

요 약

인터넷의 급속한 확산과 통신 기술의 발전이 발전함에 따라 각종 가전 제품 및 통신 장비가 개발되고 서비스의 범위와 기능도 다양해 지고 있다. 이에 따라 내장형 시스템은 시스템 고유의 기능을 지원할 수 잇는 내장형 운영체제를 필요로 한다. 본 논문에서는 리눅스의 강력한 네트워킹 기능을 이용한 리눅스 라우터의 제작 사례를 바탕으로 내장형 리눅스 시스템을 개발하고, 내장형 리눅스를 커널로 사용함에 있어 발생되는 문제점의 해결방안을 기술하고자 한다. 개발된 내장형 리눅스 라우터와 영업용 라우터와의 성능 평가결과 대등한 성능을 제공하고 있다.

Design and Implementation of Embedded Linux Router

Minkyu Joo[†]·Kyunghee Choi[††]·Jongwuk Moon[†††]
Jongsu Kim[†††]·Gihyun Jung[††††]

ABSTRACT

In this paper, we describe the issues associated with implementing embedded Linux system. As an example of embedded system, a router which utilizes the powerful networking capability of Linux is implemented and the details of porting Linux to the dedicated hardware is discussed. Several efficient methods to avoid performance degradation resulted from porting lap top computer oriented Linux to embedded system are suggested. To verify the eligibility of the method to embed Linux into standalone system and to see the performance of the implemented router, comparison data with one of the most popular routers is presented.

키워드 : Embedded Linux System, Flash File System, Router

1. 서 론

내장형 리눅스 시스템(Embedded Linux System)이란 마이크로 컨트롤러를 비롯한 하드웨어와 특정 기능을 수행하는 소프트웨어를 지원하기 위하여 운영체제로서의 리눅스를 의미한다. 내장형 리눅스의 응용 범위는 네트워 장비를 비롯하여 가전제품, PDA, 셋톱박스 등에 이르기까지 그 응용 범위가 매우 다양하다. 내장형 시스템을 위한 커널로써 pSOS, WindowsCE와 Palm OS 등의 상용 내장형 커널도 많이 사용된다. 상용 운영체제가 가격이 비싸고 커널 자체에 대한 수정이 불가능하다는 단점을 가지는 데에 반하여 리눅스는 많은 장점을 가지고 있다. 예를 들면, 상용 커널과 동일한 기능을 제공하고 있으며, 커널에 대한 소스코드의 수정이 매우 용이하고 원하는 기능만으로 구성되는 작은 크기의 커널도 만들 수 있다. 또한, 개발과정에 필요한 많은 도구 소프트웨어는 물론 응용 애플리케이션들이 공개되어 있기 때문에, 추가 소프트웨어 구입 비용이 없이도 양질의 소프트웨어를 사용할 수 있어서 개발 시간을 단축시킬 수 있다. 특히 리눅스 커널은 자체적인 네트워크 프로토콜 스택을 완벽하게 지원하고 있기 때문에 각종 통신 장비나 인터넷 디바이스 등의 운영 체제로서 매우 적합하다. [14]

리눅스 커널을 다양한 프로세서용으로 이식 작업이 진행되었지만, 여전히 특정 내장형 시스템에 활용하기 위하여는 나름대로의 이식 작업을 수행하여야 한다. 예를 들면, 리눅스가 상용 유닉스를 목표로 하고 있어 소스코드가 매우 방대하기 때문에 소스 코드 중에서 필요한 부분과 그렇지 않은 부분을 가려내어야 하는 과정이 필요하다. 또한 대부분의 내장형 시스템은 고유의 장치와 그 장치 관리 기술을 필요로 하고 있고, 디스크 등 저장 장치를 사용하지 않아 커널 실행 이미지나 루트 파일 시스템에 대한 저장 공간을 위한 해결책이나 가상 메모리와 페이징을 사용함으로 인한 스와핑 공간에 대한 해결책이 필요하다. 본 논문에서는 내장형 시스템의 대표적 예라고 할 수 있는 인터넷 종단 라우터를 리눅스를 기반으로 개발한 경험을 기술하고자 한다.

본 논문의 구성은 다음과 같다. 2절에서는 개발 환경에 대한 구성과 목표 보드에 대해 간략히 살펴보고, 목표 보드

† 준 회 원 : 아주대학교 정보통신 전문대학 대학원 정보통신 공학과
†† 정 회 원 : 아주대학교 공과대학 컴퓨터공학과 정교수
††† 준 회 원 : 아주대학교 대학원 전자공학과
†††† 정 회 원 : 아주대학교 전기전자공학부 교수
논문접수 : 2001년 10월 4일, 심사완료 : 2001년 11월 2일

의 부팅을 위한 모니터 프로그램에 대해 기술한다. 3절에서는 최소한의 커널 수정 등 이식 과정과 내장형 리눅스 시스템에서의 커널 부팅 전략과 플래쉬 메모리 사용 방안에 대해 기술하고 가상 메모리와 페이징을 사용하는 리눅스 시스템에서의 스와핑 문제에 대해서 기술한다. 4절에서는 개발된 시스템의 성능평가를 통하여 리눅스가 라우터의 운영체제로도 적합함을 보이고 5절에서 결론을 기술한다.

2. 개발 환경 및 모니터 프로그램

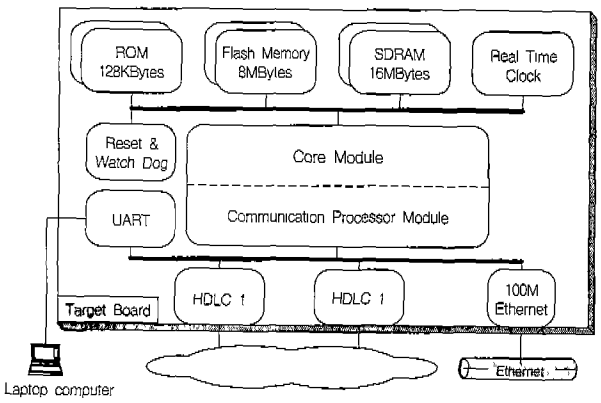
내장형 리눅스를 이용한 라우터 개발의 첫 과정은 목표 보드를 제작하고, 하드웨어 기본 초기 설정 및 디버깅을 제공하는 모니터 프로그램을 제작하는 것이다. 본 절에서는 목표 보드 및 모니터 프로그램의 개발 과정에 대해 기술한다.

2.1 목표 시스템의 기능

본 연구에서 개발한 라우터는 소규모의 회사 및 가정에서 사용할 수 있는 소형 IP 라우터이다. PPP(Point-to-Point) 프로토콜을 사용하는 전용망과 연동될 수 있으며, 로컬 네트워킹을 위해 이더넷 프로토콜을 지원한다. 라우터의 기능 강화를 위해 현재 많은 상용 라우터에서 제공하는 NAT (Network Address Translation), DHCP(Dynamic Host Configuration Protocol) 및 라우팅을 위한 프로토콜인 RIP(Routing Information Protocol) 및 OSPF(Open Shortest Path First) 프로토콜을 지원한다. 또한, FTP, Telnet, HTTP 등의 프로토콜을 위한 소프트웨어를 제공한다.

2.2 목표 시스템의 기능

내장형 시스템을 구성함에 있어 가장 중요한 선택은 마이크로 컨트롤러의 선정이다. 현재 수많은 종류의 마이크로 컨트롤러가 출시되어 있으며 본 연구에서는 통신 모듈을 컨트롤러 내부에 탑재한 범용 통신 프로세서인 Motorola의 MPC 860T를 사용하였다. MPC860T는 PowerPC 계열의 프로세서로서 내부가 CPU Core와 CPM(Communication Processor Unit)등 두 유닛으로 구성된다. CPM은 Ethernet, UART, PPP 등의 프로토콜을 지원하기 위한 별도의 RISC로 구성되어 있다. CPU Core와 통신 장치가 구분되어 있어서 CPU Core의 도움 없이도 통신 처리가 가능하여 라우터와 같은 통신 장비에 널리 쓰이고 있다. 본 연구에서는 CPM을 시리얼 콘솔을 위한 UART, 10/100 Mbps 이더넷을 위한 인터페이스, 그리고 HDLC PPP를 위한 2개의 인터페이스로 사용하였다. 그 밖의 구성으로 모니터 프로그램이 적재될 수 있는 ROM과 SDRAM, 그리고 커널 실행 이미지와 같이 중요한 데이터를 보관하는 플래쉬 메모리가 탑재되어 있다. 목표보드의 전체적 구성은 (그림 1)과 같다[7].



(그림 1) 목표 보드

2.3 개발 환경

커널 및 응용 프로그램이 목표 보드에서 실행되기 위해서는 목표 보드의 프로세서에 적합한 바이너리를 생성하는 컴파일러가 필요하다. 일반적으로 내장형 시스템을 위한 컴파일 작업은 목표 보드에서 이루어지지 않기 때문에 크로스 개발 환경을 이용한다. 본 연구에서는 크로스 개발 환경 구축을 위해 GCC(GNU Compiler collection)를 사용하였다. GCC는 매우 다양한 프로세서들을 위한 개발 환경을 제공하며, 리눅스 커널을 컴파일하는 데 있어 가장 적합한 개발 툴 중의 하나이다. 또한, 링커, 오더 등의 바이너리 유틸리티를 비롯하여 컴파일러 및 디버거가 제공된다.

유저 프로세스의 형태로 실행되는 대부분의 프로그램은 라이브러리를 필요로 한다. 라이브러리 역시 크로스 개발 환경에 필수적인 요소로써, 목적으로 하는 프로세서에 부합하도록 구축되어야 한다. 본 연구에서 사용한 라이브러리는 GNU에서 개발된 glibc를 사용하여 구축하였다.

2.4 모니터 프로그램의 개발

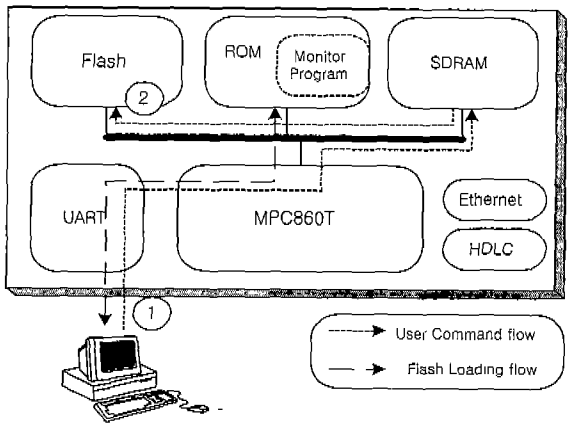
모니터 프로그램은 내장형 시스템의 초기 동작에 대한 설정과 커널 로딩을 통해 본격적으로 커널을 실행시키는 펌웨어(Firmware) 소프트웨어이다. 제작한 목표 보드가 올바른 동작을 하는 지에 대한 디버깅 목적으로도 사용된다. 모니터 프로그램의 제작은 내장형 시스템 개발에 있어 가장 우선적으로 수행되어야 한다. 개발된 모니터 프로그램은 ROM 영역에 영구적으로 저장되어 실행된다[14].

일단 작성된 리눅스 커널은 모니터 프로그램에 의해 메모리에 로딩이 되어야 한다. 모니터 프로그램이 어떤 장치로부터 커널을 메모리로 로딩하는 메커니즘은 내장형 시스템의 구성에 따라 다양하다. 본 연구에서는 플래쉬 메모리를 이용하는 메커니즘을 사용하였다. 커널을 비롯한 루트 파일 시스템 이미지, 라우터에 관련된 수 많은 설정 정보 등 변경될 수 있는 것들은 플래쉬 메모리에 저장하고, 초기 커널의 실행은 모니터 프로그램에 의해서 이루어 진다. 그 밖의 저장된 정보들은 리눅스 디바이스 드라이버를 통해 억세스 된다.

초기에 시스템을 부팅할 때 사용될 정보, 즉 로딩될 커널

이미지 및 루트 파일 시스템으로 사용될 파일 시스템 등을 위하여 모니터 프로그램에서는 간단한 로딩 기능을 제공한다. 커널이 부팅된 후 플래쉬 메모리에 대한 억세스는 유닉스 *VFS*(Virtual File System)의 블록 디바이스 드라이버를 통해 이루어 지도록 플래쉬 디바이스 드라이버가 필요하다. 이에 대한 자세한 사항은 다음절에서 다루도록 한다. 그림 2에서 볼 수 있듯이 크로스 개발환경을 통해 컴파일된 커널과 루트 파일 시스템은 모니터 프로그램을 통하여 SDRAM에 임시적으로 저장된 후 플래쉬에 저장된다.

일반적으로 호스트 컴퓨터와 내장형 시스템과의 통신은 RS-232c 시리얼 통신 방식이기 때문에 큰 용량의 데이터 전송시 에러 발생율이 높다. 이를 위해 (그림 2)에서와 같이 SDRAM을 중간 버퍼로 사용하여 메모리로의 로딩이 완료되면 비로소 플래쉬 메모리에 업데이트가 되도록 한다. 모든 수행은 모니터 프로그램에 의해 이루어지며, 그 밖에 사용자로부터 요청되는 시스템 명령이 모니터 프로그램에 의해 수행되도록 하였다.



(그림 2) 모니터 프로그램

3. 내장형 시스템을 위한 커널 이식

이 절에서는 리눅스 커널 버전 2.2.5를 바탕으로 내장형 리눅스를 이식하는 과정에 대해 기술한다. 하드웨어에 의존적인 코드 수정에 대하여 기술한 후, 내장형 시스템에 사용되는 플래쉬 메모리의 사용 전략 및 디바이스 드라이버의 구현에 대해 기술한다. 대용량의 저장 장치를 사용할 수 없는 내장형 시스템에서 발생하는 스와핑과 그에 대한 해결 방안에 대하여도 기술한다.

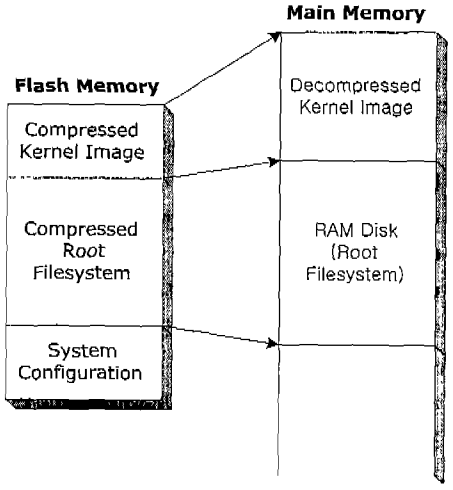
3.1 하드웨어 의존적인 코드의 이식

리눅스 커널을 구성하는 소스 코드는 커널이 가지는 그 역할 및 기능에 따라 계층적인 트리 구조로 정돈되어 있다. 따라서 특정 하드웨어에 대한 커널 이식에 상당한 편리성을 제공한다. 특히, 하드웨어에 의존적인 코드들은 *arch* 디렉터리의 하위에 프로세서의 종류에 따라 구분되어 있기 때문에 특정 시스템을 위해 수정해야 할 소스 코드들을 쉽게 파악할 수 있다. 널리 사용되는 프로세서 및 개발의 목적으로 제작된 목표 보드에 대해서는 상당 부분 이식되었기 때문에 이식 과정을 위한 시간을 상당히 감소시킬 수 있다. 그러나 본 연구를 위해 제작된 목표 보드처럼 특정 목적을 위해 제작되었다면, 하드웨어 의존적인 소스 코드의 수정은 불가피하다. 하드웨어 의존적인 소스 코드는 크게 시스템 초기화를 위한 부분과, 특정 하드웨어를 억세스 할 수 있도록 제작되는 디바이스 드라이버이다. 리눅스 커널이 기본적으로 제공하는 기능 자체를 변경하거나, 특정 기능만을 목적으로 성능향상을 꾀하고자 할 때에는 하드웨어에 의존적이지 않는 부분의 수정도 필요하다.

내장형 시스템은 특정 목적만을 위해 개발되기 때문에, 메모리나 기타 장치들의 주소 공간을 어떻게 할당하는가는 매우 중요하다. 본 연구에서 사용된 프로세서는 *PowerPC* 계열로 *Memory Mapped I/O*를 사용한다. 리눅스가 가상 주소를 사용하기 때문에 프로세서가 *Memory Mapped I/O*를 사용하는 경우 여러 주변 장치들을 커널에서 접근하도록 가상 주소 공간에 매핑되어야 한다. 예를 들면, SDRAM, Flash, 기타 여러 장치들을 사용하기 위한 메모리 메핑이 우선적으로 선행되어야 해당 장치의 사용이 가능해 지며, 이 과정은 커널이 제공하는 *ioremap*( ), *iounmap*( )과 같은 커널 API를 사용하면 된다. 다음으로 인터럽트와 예외 처리를 위한 핸들러를 작성하고 이를 등록하여 인터럽트 기반으로 동작되는 여러 장치들을 수행 가능하도록 한다.

3.2 내장형 시스템에서의 플래쉬 메모리

내장형 시스템에서 하드 디스크와 같은 대용량 저장 장치는 아니더라도 어느 정도의 정보를 저장하기 위한 저장 공간은 필요하며, 이를 위해 ROM이나 플래쉬 메모리가 사용된다. ROM은 그 크기에 한계가 있으며 빈번한 I/O가 불가능하고 실행 중 저장 용도로 사용할 수 없다. 이에 비해 플래



(그림 3) 플래쉬 메모리의 구성

쉬 메모리는 비교적 큰 용량을 저장할 수 있으며, 시스템 실행 중 읽기와 쓰기가 가능하고, 대용량 저장 장치 보다 빠른 접근 시간을 가지기 때문에 내장형 시스템의 저장 장치로 널리 사용되고 있다. 본 연구에서도 커널과 루트 파일 시스템 이미지, 라우터의 설정 정보를 저장하기 위한 저장 장치로써 8MB 크기의 *NAND* 형 플레쉬 메모리를 사용하였다.

플레쉬 메모리를 (그림 3)과 같이 특정 크기의 파티션으로 나눈 후, 상위 2MB는 압축된 커널 이미지를 저장하는 공간으로 사용하였으며, 다음의 4MB는 압축된 파일 시스템 이미지를 저장하는 공간으로 사용하였다. 마지막 2MB에는 파일 시스템을 구성하여 실행 중 *VFS* 파일 시스템의 형태로 접근될 수 있도록 하였다. 압축된 커널 이미지와 루트 파일 시스템 이미지는 앞의 2장에서 설명한 모니터 프로그램에 의해 플레쉬 메모리에 저장된다. 압축된 커널 이미지는 부팅 과정에서 압축이 풀려진 후 실행되도록 압축을 풀어주는 코드를 압축 커널의 앞부분에 압축되지 않은 형태로 존재한다. 압축을 하지 않는 상태로도 부팅이 가능하지만, 저장 공간의 절약을 위해 본 연구에서도 이 전략을 사용하였다. 저장된 압축 커널 이미지는 모니터 프로그램에 의해 부팅 시 메모리의 특정 주소 공간으로 로딩이 된다. 로딩된 압축 커널 이미지의 첫 부분이 실행되면 압축된 커널은 메모리의 다른 영역으로 압축이 해제되고 비로소 리눅스 커널이 실행된다.

리눅스 커널은 부팅 중 루트 파일 시스템을 마운트한다. 일반적인 리눅스의 경우 하드 디스크의 파일 시스템을 루트 파일 시스템으로 마운트하지만, 내장형 시스템은 하드 디스크와 같은 저장 장치가 없기 때문에 다른 장치로부터 마운트가 필요하다. 이를 위해 본 연구에서 램디스크(Ram Disk)를 사용하였다. 루트 파일 시스템을 구성하여 하나의 파일로 생성한 후 압축한 상태를 (그림 3)과 같이 플레쉬 메모리에 저장한다. 커널 부팅 시에 이 영역을 읽어 압축을 풀면서 램 디스크에 복사를 한다. 압축 해제가 완료된 후, 램 디스크를 루트 디바이스로 마운트 한다. 램 디스크를 사용함으로써 빠른 억세스 시간을 보장할 수 있다는 것은 장점이지만, 램 자체가 휘발성이기 때문에 시스템이 구동 중에 변경되는 정보에 대해서는 영구적인 저장이 불가능 하다는 단점을 가진다. 이런 단점을 극복하기 위해 플레쉬 메모리의 마지막 파티션을 이용한다. 이 영역은 자체적으로 하나의 파일 시스템으로 마운트 될 수 있도록 구성하여 시스템의 운영 중에 하드 디스크처럼 I/O가 가능하다. 이를 위해 커널은 *VFS*를 지원하는 플레쉬 디바이스 드라이버를 제공해야 한다. 플레쉬 디바이스 드라이버를 *VFS* 형태로 제공함으로써 얻을 수 있는 장점은 시스템의 운영 중에도 모니터 프로그램의 도움 없이 플레쉬 메모리의 커널 저장 영역과 루트 파일 시스템 저장 영역을 접근할 수 있다는 점이다. 이를 이용하면 모니터 프로그램을 통하지 않고 커널과 루트 파일 시스템을 플래쉬 메모리 영역에 로딩할 수 있다. 모니터 프로그램은 UART 프로토콜을 사용하기 때문에 I/O의 속도가 매우 느리다는 단점을 가지지만, 이더넷 환경을 통한 데이터를 전송하고, 플래쉬 억세스를 위한 *VFS*를 이용한다면 이런 단점을 극복할 수 있다.

플레쉬 메모리는 유닉스 *VFS*의 구조에 따라 블록 디바이스로 구분된다. 본 연구에서 사용한 *NAND*형 플레쉬 메모리는 읽을 시에는 512 Bytes, 쓸 때는 8 KBytes 블록 단위로 억세스 해야 한다. 특히 쓰기 작업을 할 때는 해당하는 8 K 블록을 반드시 지워야 하는 단점이 있다. 리눅스 블록 디바이스는 저수준 I/O의 단위로 512 바이트를 사용한다. 이는 하드 디스크의 섹터 단위에서 기인하는 것으로 이를 플래쉬 메모리에 적용하면, 512 바이트를 플레쉬 메모리에 쓰기 위해서는 8K 바이트를 읽어 해당하는 512 바이트를 업데이트 한 후 다시 8K 바이트에 대해 쓰기를 수행해야 하므로 쓰기 작업에서 상당한 성능 저하를 가져온다. 본 연구에서는 이러한 단점을 극복하기 위해 다음과 같은 플레쉬 디바이스 드라이버의 메커니즘을 사용하였다.

플래쉬 메모리의 억세스를 위한 각 시간을 다음과 같이 정의할 경우,

$$Tr : Flash\ read\ time,\ Te : Flash\ erase\ time,\ Tw : Flash\ write\ time \quad (1)$$

식 (1)에 의해 플래쉬 단위 섹터의 내용을 업데이트 하는데 걸리는 시간 Ts는 아래와 같이 정의된다.

$$Ts = Tr + Te + Tw + c \ (\ c : Constant\ ) \quad (2)$$

만약, 플래쉬 메모리 내의 8KBytes 블록 내부에서 연속적인 데이터 쓰기가 수행될 때 최악의 경우 16*Ts*의 시간이 소모된다. 커널 이미지나 루트 파일 시스템을 플레쉬 디바이스 드라이버를 사용하여 업데이트와 같이 연속적인 공간에 대한 억세스가 이루어 질 경우, 소비되는 시간은 억세스하는 섹터의 개수에 따라 그 오버헤드가 선형적으로 증가됨을 볼 수 있다. 이를 극복하기 위해 연속적인 구간에 엑세스가 이루어 질 경우에 디바이스 드라이버의 버퍼링을 통해 그 성능을 향상시킬 수 있다. 즉, 플레쉬 디바이스 드라이버는 버퍼링을 통해 8KBytes 단위의 버퍼링을 수행하고 8KBytes 버퍼가 가득찰 때만 실제 플레쉬 쓰기를 수행한다면 최고 94.65%의 실행 시간을 단축시킬 수 있으며 평균적으로 88.78%의 시간 단축 효과를 볼 수 있다. 이 메커니즘은 앞에서 설명한 플레쉬 메모리의 사용 용도 중 커널과 루트 파일 시스템의 저장과 같이 플래쉬 영역에 대한 연속적인 억세스를 수행하는 예에 매우 적합한 방법이다.

디바이스 드라이버의 구성은 플레쉬 영역 중 마운트 된 영역을 접근하려 한다면, 일반적인 I/O 루틴을 수행하도록 하고 그렇지 않은 경우 즉, 커널 저장 영역이나 루트 파일 시스템 저장 영역에 대한 쓰기 작업은 버퍼링을 통한 억세스가 되도록 하였다. 이런 플레쉬 억세스 타입에 대한 결정은 디바이스 파일의 Minor number로 구분할 수 있다. 버퍼링 메커니즘이 수행될 수 있는 플레쉬 영역은 파일 시스템이 아닌 오직 하나의 데이터가 저장된 영역이기 때문에 항상 데이터의 첫 블록부터 끝 블록까지 연속적인 접근만이

유효하다. 쓰기 작업 역시 첫 블록부터의 쓰기가 이루어지기 때문에 512 바이트 블록 16개씩 버퍼링을 하여 8K 바이트가 채워지면 비로서 쓰기를 수행하도록 한다. 이 경우 8K 바이트가 채워지지 않는 마지막 8K 블록에 대한 처리가 필요하게 되는데 이는 주기적으로 수행되는 커널 쓰레드를 통해 해결하도록 한다. 커널 쓰레드는 보통의 경우 수면 상태를 유지하다가 플래쉬에 대한 억세스가 이루어 지면 커널에 의해 수면 상태에서 깨어난다. 수면 상태에서 깨어난 후 일정 시간마다 실행이 되는데, 매 실행 시 플래쉬에서 사용되는 버퍼의 마지막 억세스 시간과 현재 시간과의 차이를 검사한다. 이 시간차가 특정 값 이상이면, 플레쉬에 대한 마지막 버퍼라 간주하고 플래쉬에 쓴다. 시간에 대한 기본값은 실험을 통해 얻어질 수 있으며 본 연구에서는 대략 500ms이다.

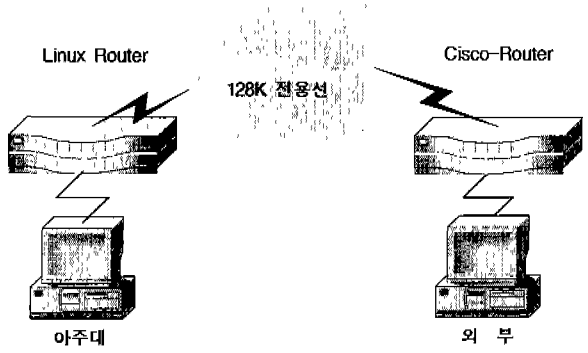
3.3 메모리 스와핑

리눅스는 가상 메모리와 페이징 기법을 사용하는 운영체제로서 스와핑 메커니즘을 필요로 한다. 리눅스는 항상 충분한 자유 메모리를 유지하기 위해서 커널 메모리 관리 모듈이 실행될 때마다 유저 프로세스가 사용하는 메모리 영역을 스왑하려고 시도한다. 스왑은 일정한 자유 메모리가 존재하지 않을 때 유저가 사용하는 영역에 대해 강제적으로 수행이 되거나 혹은 주기적으로 실행되는 커널 쓰레드에 의해 이루어진다. 두 경우 모두 하드 디스크와 같은 물리적인 저장 장치의 스왑 파티션에 저장된다. 앞에서도 언급한 바와 같이 내장형 시스템에서는 이러한 목적으로 사용될 수 있는 저장 장치가 존재하지 않는다. 따라서 리눅스를 내장형 시스템의 커널로 사용하기 위해서는 스왑에 대한 해결이 필요하다.

첫번째 해결 방법은 커널 수정을 통한 스왑을 영구적으로 방지하는 것이다. 일반적으로 내장형 시스템은 그가 목적으로 하는 기능만을 수행할 정도의 하드웨어로 구성된다. 즉, 내장형 시스템에서 수행되는 커널을 비롯한 여러 프로세스들은 물리적인 메모리 크기의 범위 내에서 돌아갈 수 있도록 구성하기 때문에 메모리 부족 현상에 대한 처리는 필요하지 않을 정도로 물리적인 시스템을 구성한다. 따라서 스왑을 수행하는 모듈을 커널에서 제거함으로서 스왑 문제를 해결할 수도 있다. 즉, 주기적으로 수행되는 스왑 커널 쓰레드를 초기에 생성을 하지 않도록 수정함으로써 해결한다. 두 번째 해결책으로 어떠한 형식으로든 스왑 영역을 확보하여 어느 정도의 메모리는 스왑 아웃될 수 있도록 커널을 수정하는 방법이다. 스왑 영역으로 사용될 수 있는 저장 장치는 램외 특정 영역, 플레쉬 메모리, NFS를 통한 억세스 등이 있다. 본 연구에서는 첫 번째 방법을 사용하였다.

4. 성능 평가

본 논문에서 개발한 라우터에 대한 성능 실험을 위한 구성은 (그림 4)와 같다. 두 시스템 사이에 128K 속도의 전용 회선을 설치하고 양 단에 라우터를 설치한다. 라우터 사이의 네트웍 인터페이스는 V.35 DSU를 사용하였다. 라우터 뒷단에는 PC를 물려 PC-to-PC 사이에서의 전송 속도를 측정한다. 본 연구에서 개발한 라우터의 성능을 비교하기 위해 가장 널리 사용되는 시스코 라우터를 이용하였다. 라우터간의 프로토콜은 Cisco-HDLC를 사용하였다.



(그림 4) 라우터 성능 측정 환경

<표 1>은 FTP 프로토콜을 이용하여 양단의 PC에서 FTP를 이용하여 1MBytes, 10MBytes의 파일을 전송하는데 걸리는 시간에 대한 비교이다. <표 1>에서 보는 바와 같이 내장형 리눅스를 이용한 라우터의 성능은 일반 상용 라우터와 거의 균등한 처리 속도를 보여주고 있다.

〈표 1〉 FTP를 이용한 파일 전송

(단위 : sec)

타 입 / 회 수	Cisco-Cisco		Cisco-Linux	
	1MB	10MB	1MB	10MB
1	74.43	648.94	75.03	652.22
2	74.31	648.85	74.97	652.12
3	74.39	648.90	75.08	652.07

<표 2>는 양단의 PC 사이에서 ICMP 프로토콜을 이용하여 양단의 round-trip delay을 측정한 결과이다. 측정은 64KB, 1024KB의 크기를 가지는 ICMP 패킷을 500번 수행한 후 걸린 시간의 평균치이다.

〈표 2〉 Round-trip delay

(단위 : ms)

타 입 / 회 수	Cisco-Cisco		Cisco-Linux	
	64KB	1024KB	64KB	1024KB
1	24	148	24.6	148
2	24	148	20	148
3	24	148	20	148

5. 결론 및 향후과제

본 논문에서는 내장형 시스템의 한 예인 라우터에 리눅

스를 이식하고 그 개발 과정에 대해 논의하였다. 시스템 개발을 위해 우선적으로 구축되어야 할 개발 환경에 대해 간단히 살펴보았으며, 개발된 하드웨어 시스템의 디버깅 및 커널 로딩을 비롯한 시스템 의존적인 기능을 수행하는 모니터 프로그램에 대해 논의하였다. 특히, 플레쉬 메모리 등 내장형 시스템에서 자주 사용되는 장치들을 리눅스 커널에 응용하여 내장형 시스템의 기능을 강화시키고, 그 성능을 향상시키기 위한 방안을 제시하였다. 스왑핑 등 리눅스를 내장형 시스템의 커널로서 사용함에 있어 발생하는 문제점 및 그 해결 방안을 제안하였다. 마지막으로 본 연구에서 개발된 라우터의 성능을 기존에 개발되어 사용되고 있는 라우터와의 성능을 분석하고 비교하였다. 그 결과는 네트웍 장비 회사에서 나오는 상용 라우터에 비하여 크게 손색이 없을 정도이다. 이는 리눅스를 이용한 내장형 시스템의 응용이 매우 광범위하게 사용될 수 있음을 의미한다.

참 고 문 헌

[1] Bollinger, T., "Linux in practice : an overview of applications," IEEE software, Vol.16 Issue 1, Jan-Feb. 1999.

[2] Daniel P. Bovet, Marco Cesati, "Understanding the Linux Kernel," 1st Ed., O'Relly, 2001.

[3] Shahid H. Bokhari, "The Linux Operating System," IEEE computer, Vol.28, No.8, 1995.

[4] S.Keshav, R.Sharma, "Issues and trends in router design," *IEEE Communications Magazine*, May, 1998.

[5] David A. Rusling, "The Linux Kernel," Linux Documentation Project, 1998.

[6] Motorola, "MPC860 PowerQUICC," User's Manual.

[7] S. Radhakrishnan, "Linux-Advenced Networking Overview Version 1," Information and Telecommunications Technology Center, 1999.

[8] M. Labrador, S. Banerjee, "Packet Dropping Policies for ATM and IP Networks," IEEE Communications surveys, Vol.2 No.3, 1999.

[9] W. Werner, J. Salim, A. Kuznetsov, "Differentiated Services on Linux," EPFL ICA, 1999.

[10] Glenn Herrin, "Linux IP Networking," TR00-04, 2000.

[11] C. Kever, "Linux Kernel Hash Table Behavior : Analysis and Improvements," Proceedings of the 4th Annual Linux Showcase & Conferene, 2000.

[12] R. Riel, "Page replacement in Linux 2.4 memory management," USENIX Annual Technical Conference, 2001.

[13] M. Joo, K. Choi, et al, "Development of Embedded Linux Router," Proceedings of the 28th KISS Sping Conference, 2001.

[14] http : //linux-mm.org/.

주 민 규

e-mail : zzu@cesys.ajou.ac.kr

2000년 아주대학교 정보 및 컴퓨터 공학 졸업(학사)

2000년~현재 아주대학교 정보통신 전문대학 정보통신 공학 전공(공학 석사과정)

관심분야 : 임버디드 시스템, RTOS, 통신 프로토콜, 인터넷 QoS

최 경 희

email : khchoi@madang.ajou.ac.kr

1976년 서울대학교 사범대학 수학교육과 (학사)

1979년 불란서 그랑데꼴 ENSEIIHT, 정보공학 및 응용수학(석사)

1982년 불란서 Univ. of Paul Sabatier, (박사)

1991년~1991년 불란서 렌느 IRISA 연구소 교환교수

1982년~현재 아주대학교 공과대학 컴퓨터공학과 정교수

관심분야 : 운영체제, 분산 처리, 실시간 시스템

김 종 수

e-mail : promise@csl.ajou.ac.kr

2000년 아주대학교 전자공학과 졸업(학사)

2000년~현재 아주대학교 전자공학과 (석사과정)

관심분야 : 초고속 LAN 통신망, 임버디드 시스템, RTOS 등

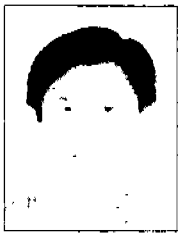
문 종 욱

e mail : lache@madang.ajou.ac.kr

2000년 아주대학교 전자공학부 졸업(학사)

2000년 현재 아주대학교 전자공학부 석사 과정

관심분야 : 실시간 운영체제, 임베디드 시스템, 실시간 패킷 분석



정 기 현

e-mail : khchung@madang.ajou.ac.kr

1984년 서강대학교 전자공학과 졸업(학사)

1988년 미국 Ilinois주립대 EECS 졸업 (석사)

1990년 미국 Purdue대학 전기전자공학부 졸업(박사)

1991년~1992년 현대반도체 연구소

1993년~현재 아주대학교 전기전자공학부 교수

관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어 및 실시간 시스템 등