PowerPC에 기반한 내장형 시스템을 위한 리눅스의 이식

강경태° 김태웅 박상수 신현식 장래혁
서울대학교 컴퓨터공학부
{nicola, twkim, sspark}@cselab.snu.ac.kr, shinhs@snu.ac.kr,
naehyuck@snu.ac.kr

Porting Linux for PowerPC-based Embedded Systems

Kyungtae Kang° Taewoong Kim Sangsoo Park Heonshik Shin Naehyuck Chang
School of Computer Science and Engineering, Seoul National University

요 약

인터넷 등 통신의 발달과 맞물려 컴퓨터, 이동전화, 셋톱박스(STB), 디지털TV, 개인휴대단말기(PDA) 등 정보단말기의 네트워크화가 급진전되면서 「내장형 리눅스」가 최근 정보기술(IT) 분야의 새로운 키워드로 부상하고 있다. 내장형 시스템(Embedded System)은 특정 기능을 수행하도록 설계된 초소형 운영체제를 탑재해 기능을 최적화한 것으로, 컴퓨팅 기능을 지닌 모든 장비는 내장형 시스템의 적용분야가 된다. 내장형 운영체제(OS)로는 리눅스, 윈도CE, 팜OS 등이 인기를 끌고 있으며, 최근 들어 리눅스를 이용한 내장형 시스템 개발이 활기를 띠고 있다. 본 논문에서는 특히 Motorola사의 MPC860 마이크로 프로세서를 이용한 내장형 시스템 플랫폼에 리눅스를 이식한 사례를 중심으로 리눅스 이식의 방향을 제시하고 있다.

1. 서론

내장형 리눅스는 리눅스의 장점을 유지하면서 시스템을 구성할 수 있다. 즉 소스가 공개돼 있어 저가로 시스템을 구성할 수 있으며 네트워크 운영자 입장에서도 운용이 용이하며 유연성이 높다. 한마디로 윈도우보다 안정적이고 유닉스보다 저렴하다는 장점이 있다.

또한 실시간 속성을 포함한 리눅스의 경우 신속, 정확하고 지속적인 정보처리가 가능하다. 따라서 리눅스를 탑재한 컴퓨터는 낮은 기종에서도 패킷의 손실 없이 고속으로 데이터를 전송할 수 있다[6][7]. 이외에도 내장형 리눅스는 리눅스 마이크로 커널을 활용할 경우 내장형 시스템의 메모리 요구량을 최대 10분의 1까지 줄일 수 있다. 이처럼 리눅스의 장점과 내장형 시스템의 요구조건이 부합되면서 통신, 인터넷 분야에서 내장형 리눅스의 사용이 확산되고 있다.

본 논문에서의 목표 보드는 Motorola사의 MPC860을 사용한 내장형 시스템이다. 물론 리눅스는 이미 파워 매킨토시를 비롯하여 소수의 PowerPC 기반 보드에 이식되어 있다. 그러나 동일한 PowerPC 코어라 하더라도 아키텍쳐에 따라 서로 다른 설정이 필요하며 동일한 아키텍쳐에서도 부팅 방법에 따라서 다르게 운영 체제를 구성해야 한다.

본 연구에서는 이를 위해 부트 로더 및 커널을 수정하였으며 구현에 대한 자세한 설명 보다는 전체적인 작업과 관련된 문제에 대한 경험에 초점을 맞추었다. 여기에는 크로스 개발 플랫폼 구성, 부팅 시퀀스 디자인, 커널 수정, 실행 이미지와 루트 파일 시스템 생성등에 대한 내용이 포함된다. 이를 위하여 목표 보드에 대한 아키텍쳐를 간단히 살펴보고 구현한 사례들을 들어 설명하도록 하겠다.

2. 전체 시스템 구성

구현된 시스템의 전체 구성은 그림 1과 같다. 메인 CPU로 PowerPC코어를 탑재한 MPC860을 사용하였으며 내부 버스와 외부 버스에 모두 50MHz의 클럭을 사용하였다. 본 프로세서는 시리얼 통신이나 이더넷 통신을 위한 콘트롤러를 비롯하여 DRAM이나 플래시 메모리와 같은 메모리 장치에 대한 메모리 콘트롤러를 포함하고 있다. 그러나 부동 소수 연산을 위한 FPU(floating point unit)를 제공하지 않는다. 따라서 목표 보드는 그림 1과 같이 별도의 콘트롤러 없이 외부 장치와의 직접적인 연결이 가능한 반면에 부동 소수 연산을 위한 소프트웨어적인 지원이 필요하다.

본 시스템에서는 512KB 의 DIP타입 플래시 메모리가 부트롬으로 사용되었으며 메인 메모리로는 64MB SDRAM을, 커널 이미지의 저장 장치로 8MB 용량의 플래시 메모리를 사용하였다[1][3]. 그리고 주변 PCI 장치와의 통신을 위한 PCI to Processor 브리지로 Tundra 사의 QSpan 칩을 사용하였다.

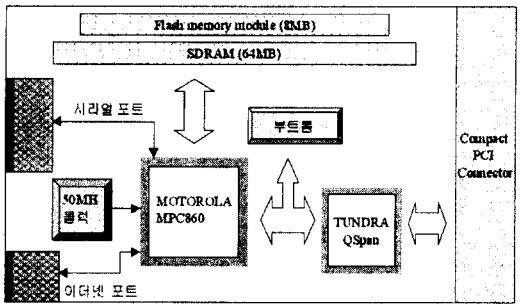


그림 1. 목표 보드

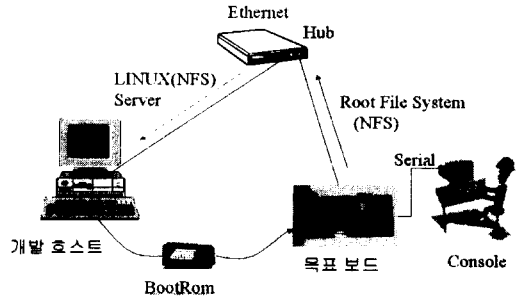


그림 2. 개발 환경

3. 리눅스 이식

그림 2는 목표 보드의 개발 환경을 보여준다. 개발 호스트는 레드햇 6.1을 탑재한 펜티엄 시스템이며 디버깅을 위해 시리얼 콘솔을 사용했다. 개발의 편의를 위하여 우선 NFS를 사용하여 루트 파일 시스템을 구축하였으며 최종적으로 램 디스크를 이용하였다.

3.1 크로스 개발 환경

커널과 기타의 응용 프로그램을 x86과 같은 개발 호스트에서 PowerPC 용으로 생성하기 위해서 크로스 개발 환경을 사용한다. 일반적으로 목표 보드보다 상대적으로 빠른 개발 호스트를 사용 할 경우 이는 매우 효과적일 수 있다. 이러한 크로스 개발 환경의 구축은 크게 binutils과 gcc 컴파일러 및 glibc의 설치로 요약된다. 본 실험에서는 이를 위해 gnu 컴파일러인 gcc 버전 2.95.2를 사용하였으며 binutils의 경우 2.9.5버전을 그리고 2.1.3 버전의 glibc를 사용하였다.

일반적인 설치 과정은 1. 환경 설정 2. 컴파일 을 따르며 환경 설정시 타겟을 powerpc-linux로 설정해 주도록 한다[5]

3.2 모니터 프로그램 및 부트 로더

크로스 개발 환경에서 구성한 컴파일러를 이용해 우선 모니터 프로그램과 부트 로더를 컴파일하고 이를 Rom에 프로그램 한 후 실제 보드의 동작 여부를 체크해 보아야 한다. 물론 모니터 프로그램 없이 바로 부팅을 시도 할 수도 있으나 하드웨어의 정상 동작을 우선 체크하는 것이 바람직하다. 부트 로더에는 시스템이 부팅시 처음으로 수행되는 루틴이 포함 되는데 이 루틴에서 실제 목표 하드웨어에 의존적인 내부 레지스터를 초기화 한다. 이밖에 예외 상황시 발생하는 이벤트에 대한 처리를 위해 다양한 예외 처리 벡터들을 정의하고 주변 장치에 대한 메모리 공간 매핑을 한다. 마지막으로 스택 포인터위치를 초기화 하고 로더의 주 함수로 분기한다.

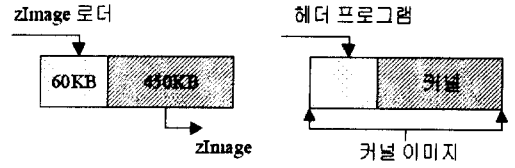
리눅스 이미지 로더의 주 함수에서는 시리얼 모니터 프로그램을 구동하기 위한 시리얼 드라이버를 초기화 해준다. 이 초기화 과정에는 CPU내의 시리얼 콘트롤러에 대한 레지스터 초기화 설정을 비롯하여 실제 하드웨어에서 사용하는 시리얼 포트의 매핑이 포함된다. 시리얼 드라이버가 초기화 된 후 시리얼 콘솔을 이용한 기본적인 시스템 점검과 디버깅이 가능하며 이후에 비로서 실제 리눅스를 부팅시키기 위한 위한 루틴을 호출하게 된다.

리눅스 부팅 루틴은 압축된 커널 이미지(zImage)의 ELF 헤더를 지정된 위치(램 영역)에 복사한 후 ELF 바이너리 헤더 정보로부터 실제 커널 프로그램의 시작 위치를 얻어 커널 이미지의 로딩을 시작한다.

커널의 로딩은 다양한 방법으로 수행할 수 있는데 크게 부트롬에 적재된 커널 이미지를 통한 방법과 온보드에 있는 플래시 메모리를 통한 방법으로 나누어 볼 수 있다. 일반적으로 부트롬은 512KB가 넘지 않는 DIP타입의 플래시 메모리를 사용하므로 전자의 방법을 사용 할 때는 커널 이미지의 크기를 작게 만들어야 한다는 제약이 따른다. 본 실험에서는 이를 위해 모니터 프로그램 및 부트 로더를 약 60KB 정도의 크기로 구성하였으며 커널 이미지의 크기를 약 450KB가 되도록 구성 하였다. 따라서 부트롬의 모습은 그림 3-(a) 와 같다. 부트롬을 구성할 때는생성한 ELF 이미지를 부트롬에 프로그램 가능한 바이너리로 전환 시킨 후 이를 다시 Motorola의 s-record 포맷으로 바꾸어 프로그램 하도록 한다.

후자의 방법을 사용할 때는 상대적으로 용량이 큰 플래시 메모리를 이용할 수 있기 때문에 커널 이미지의 용량에 크게 영향을 받지 않으며 이 경우 부트 로더는 플래시 메모리의 영역에서 커널 이미지를 로딩하게 된다.

커널 이미지는 그림 3-(b)와 같이 헤더 프로그램과 압축된 커널로 이루어져 있으며 헤더 프로그램이 이 압축된 커널을 로딩하는 역할을 한다.



(a) 부트롬 구성 (b)zImage 구성

그림 3. 로더의 구성

위에서 설명한대로 커널의 로딩은 2개의 분리된 로더를 이용하여 이루어진다. 하나는 커널 이미지 로더이고 나머지 하나는 리눅스 커널 로더라 부른다. 이미지 로더는 시스템이 부팅되면서 수행되는 필수적인 하드웨어 초기화를 수행하며, 롬에서 램의 영역으로 리눅스 커널 이미지를 옮긴다. 이어서 수행되는 커널 로더는 더 많은 하드웨어 초기화를 담당하며 동시에 리눅스 커널 부팅을 위한 환경을 셋업한다

일반적으로 이 커널 로더는 아키텍처에 종속적이기 때문에 /arch/"아키텍처"/* 디렉토리에 위치한다. 본 시스템의 경우도 마찬가지로 /arch/ppc/mbxboot/ 디렉토리에 커커널 로더가 위치한다. 커널 로더는 커널에서 사용하는시리얼 드라이버를 새로 초기화 하고 메모리 크기와 시스템

클럭 및 보드율(Baud Rate) 과 같은 하드웨어 정보를 환경 변수로 전달 받아 이를 자료구조에 저장한다. 이 저장된 정보들을 이용하여 커널 로더는 커널을 메모리상에 배치하고 압축된 커널 이미지를 하위 번지에 푼 후 커널의 진입점으로 분기한다.

3.3 커널 수정

리눅스의 소스 트리 구조는 정형화되어 있기 때문에 이식성이 높다. 이 중 특정 프로세서에 의존적인 코드는 /arch 디렉토리에 존재한다. 따라서 리눅스를 특정 프로세서에 이식한다 함은 상당 부분 이 디렉토리를 수정함을 의미한다. 본 연구에서 사용한 MPC860 프로세서의 경우에는 이더넷 프로토콜을 지원하는 콘트롤러를 프로세서 내에 포함하고 있기 때문에 위에 언급한 바와 같이 /arch 디렉토리에 있는 드라이버를 수정함으로써 이더넷 초기화를 수행할 수 있다. 또한 시리얼 초기화와 마찬가지로 이더넷을 위해 사용한 포트 정보를 커널에 전달해 주어야 한다. 마지막으로 시스템이 사용할 IP-주소 와 게이트웨이 주소 및 넷마스크 주소등의 정보를 커널의 소스에 명시적으로 지정해 준다.

커널 루틴을 수행하면서 다양한 장치에 대한 초기화가 이루어지는데 본 연구에서는 이중 PCI 관련 루틴을 수정하였다. 리눅스 PCI 드라이버는 0번 PCI 버스부터 PCI 장치를 탐색하여 찾아낸 장치에 대해 해당 자료구조의 연결 리스트를 만들어 시스템의 배치도를 작성한다. 또한 각 장치에 대한 I/O 영역과 메모리 영역을 바이오스 함수를 이용해 할당한다. 이 바이오스 함수를 이용해 특정 슬롯에 연결된 장치의 설정 헤더에 접근해 장치의 제작자 식별자와 장치 식별자를 얻어올 수 있는데 이 식별자를 검사하여 해당되는 드라이버를 호출함으로써 플러그 앤 플레이를 구현 할 수 있다[2].

위에서 설명한 PCI 루틴들을 이용하여 본 실험에서는 광채널 인터페이스를 지원하는 어댑터 보드를 설정하고 이를 바탕으로 대용량의 디스크 저장 장치를 연결한 대용량 저장 서버를 구현하였다. 또한 광채널 어댑터를 수용하는 디바이스 드라이버를 수정하여 목표 보드에 부합하는 코드를 생성하였으며 신뢰성과 성능을 향상시킨 RAID(redundant arrays of inexpensive disks) 를 사용하기 위하여 MD 드라이버를 수행하였다.

PCI 장치에 대한 초기화가 끝난 후에는 램디스크 드라이버와 네트웍 장치 드라이버를 구동하고 마지막으로 루트 파일 시스템을 마운트한 뒤 명령어 쉘을 수행함으로써 부팅이 마무리된다.

본 연구에서는 커널을 생성 하면서 PowerPC와 관련된 커널 버그를 몇 가지 수정하였다. 특히 캐시 메모리와 관련된 버그는 칩 자체의 결함과 관련되어 있기 때문에 소프트웨어적인 처리를 필요로한다[4].

3.4 파일 시스템

많은 내장형 시스템에는 디스크 또는 파일 시스템이 없다. 사실 리눅스를 실행시키는 데는 어느것도 필요치 한다. 물론 간단한 응용 프로그램의 경우는 커널과 함께 컴파일 되어 부팅할 때 하나의 이미지로 올릴 수 있으나 이러한 방법은 시스템의 융통성을 떨어뜨린다. 이러한 단점을 극복하기 위해 파일 시스템을 사용한다.

파일 시스템은 일반적인 디스크 드라이브, 플래시 메모리, 램디스크 등에 위치할 수 있다. 또한 네트웍을 지원하는 시스템의 경우에는 NFS(Network File System)을 사용할 수도 있다. 본 연구에서는 NFS와 램디스크, 그리고 스카시 하드 디스크를 이용한 파일 시스템을 구현하였으며 이중 NFS와 램 디스크에 대해서 설명을 하겠다.

(a) NFS (b) Ram Disk

그림 4. 루트 파일 시스템

그림 4-(a)는 NFS를 루트 파일 시스템으로 사용 했을때의 구성도의 모습이다. NFS를 사용할 경우에는 네트웍으로부터 유용한 응용 프로그램을 적재해서 실행하는 것이 가능하다. 그러나 빠른 수행을 전제로 하는 응용에는 적합하지 않다는 단점이 있다.

램디스크를 사용하기 위해서는 루트 이미지를 준비해야 한다. 이 방법은 커널이 실행된 후 램에 초기 루트 파일 시스템을 마운트 하도록 선택하며 이를 위해 크로스 개발 플랫폼상에 디스크를 만들어 EXT2 파일 시스템을 생성한다. 그리고 /etc, /dev, /bin 과 같은 서브 디렉토리를 만들며 필요한 바이너리와 라이브러리, 장치 파일 등을 램 디스크 안으로 복사해 넣는다. 이 램 디스크를 압축하여 ramdisk.gz을 얻어낸 후 이를 포함한 커널을 새로 생성한다[7] (그림 4-(b)참조). 램디스크를 사용할 경우 고속으로 파일 시스템을 접근 할 수 있으므로 프로그램의 빠른 수행이 가능하나 램 영역에 만들어지기 때문에 파일 시스템을 작고 간결하게 만들어야 한다는 제약이 따른다.

4. 결론

본 연구에서는 리눅스를 이식하기 위해 펜티엄 PC를 이용해 개발 환경을 설치하고 부트 로더를 개발 하였으며 커널 루틴의 일부를 수정하였다. 또한 이식된 리눅스에서 효율적인 응용 프로그램 개발을 위해 루트 파일 시스템을 구성하였다. 이러한 환경에서 Dhrystone 벤치 마크 프로그램을 비롯해 vi, gcc, gdb 등의 응용 프로그램을 안정적으로 실행 하였으며 본 실험의 결과물인 대용량 저장 서버가 필요로 하는 디스크 입출력을 효율적으로 수행하였다.

내장형 리눅스를 이식하기 위해서는 시스템의 특성과 구성 요소 등을 잘 파악하여 하드웨어 초기화를 최적화하는 것이 무엇보다 중요하며 이를 위해서는 먼저 프로세서에 대한 이해가 선행 되어야한다.

참고문헌

[1] Motorola, "Power QUICC Manual"
[2] David A Rusling. "The Linux Kernel" chapter 6. PCI
[3] http://www.motorola.com/SPS/PowerPC/
[4] Linux on PowerPC embedded system Mailing List, "http://lists.linuxppc.org/lists/linuxppc-embedded"
[5] http://www.ppc.kernel.org
[6] Y.C. Wang and K.J.Lin. "Enhancing the real-time capability of the Linux Kernel.", RTCSA'98, Hiroshima, Japan, OCT 1998
[7] Michael Barabanov, "A Linux-based Real-Time Operating System."
[8] Bollinger, T. "Linux in practice: an overview of applications" IEEE Software , Volume: 16 Issue: 1 , Jan.-Feb. 1999