리눅스를 이용한 실시간 시스템에서의 디바이스 드라이버 구현 - Implementation of Device Driver in Embedded system using Linux -

최용식* 이동현, 이상락, 신승호
Choi Youn-Sik, Lee Dong-hyun, Lee Sang-lak, Shin seng-ho

요 약

Real-Time Linux를 이용하여 실시간 운영체제가 요구하는 특성과 요구조건을 분석하고 이러한 요구조건에 부합하도록 리눅스를 하드웨어에 이식하고 하드웨어에 이식하기 위한 방법을 제시하였다. 다른 상용 실시간 운영체제(RTOS)와는 달리 리눅스는 특정 하드웨어를 지원하기 위한 별도의 개발환경을 제공하지 않는다. 이에 개발환경을 구축하고 부트로더를 개발하기 위해 목표 시스템에 부합하도록 리눅스 커널을 이식하였다. 또한 응용 개발의 유연성을 제공하기 위하여 램디스크를 이용한 파일 시스템을 지원하도록 하였으며, GPIO(general purpose I/O)를 통한 디바이스 드라이버를 제작하는 등의 실험을 통해 시스템의 안정성을 검증하였다. 실험에서는 StrongArm SA1110 마이크로프로세서를 이용하였으며 이 실험을 통해 실시간 운영체제로서의 리눅스의 활용 가능성을 확인하였다.

1. 서론

급속도로 발전하는 정보화 사회로의 이행으로 인하여 현대 사회는 디지털화 광대역화로 변하고 있다. 인터넷 사용자의 극격한 증가와 기술의 진보로 인하여 기간망의 비약적인 발전과 고속 가입자 망은 높은 증가세를 보이고 있다. 현재 Post-PC(Extended PC)분야의 발전은 150% 이상씩 성장률을 보이고 있다. Post-PC 형태의 시스템을 몇가지 살펴보면 다음과 같다. 홈오토메이션, 모바일 디바이스, 오디오, 자동항법, 디지털 이미지 시스템 등으로 분류해서 살펴 볼 수 있다. 여러 다양한 시스템들은 임베디드 시스템(Embedded System)의 형태로 구성되며, 이러한 임베디드 시스템의 연구의 중요성이 높아지고 있다. 그러나 현재 임베디드 시스템은 많은 비용이 요구되며 표준이 없고 서로 다른 하드웨어 환경

1 본 연구는 한국과학재단 지정 인천대학교 멀티미디어 연구센터의 지원에 의한 것입니다.
* 인천대학교 공대대학 컴퓨터공학과

에서 작동함으로써 주변 장치 드라이버에 대한 지원 여부 또한 불투명하다.

따라서 여러 다양한 욕구에 적극적으로 대처하고 범용적인 개발환경을 제공하는 것이 앞으로의 가장 큰 쟁점이 될 것이다. 이에 저렴한 개발비용을 가지면서 범용 개발환경을 제공할 수는 운영체제의 필요성이 절실하게 요구되고 있다. 이에 주목받고 있는 시스템이 Real-Time Linux이다.

Real-Time Linux는 커널 소스가 공개되어 있으며, 운영체제의 안정성과 신뢰성이 검증되었다. 또한 개발 도구가 모두 개방되어 있기 때문에 개발비용이 적게 들어 경쟁력을 높일 수 있다.

본 연구에서는 StrongARM SA-1110 마이크로프로세서를 이용한 목표 보드(target board)에 공개된 리눅스 커널을 분석하여 이식(porting)함으로써 Real-Time Linux를 이식하고, 리눅스 GPIO(general-purpose I/O) 디바이스 드라이버를 구현하였으며, 구현된 디바이스 드라이버가 목표 보드 상에서 안정적으로 동작되는지 실험함으로써 인터넷과 연결을 하기 위한 디바이스 드라이버를 구현하는데 있다.

2. 관련 연구

2.1 실시간 시스템의 현황

전자 제품 및 컴퓨터 시스템의 기능이 복잡해지고 상호 연관됨에 따라 시스템의 제어에 마이크로프로세서 및 운영체제를 삽입하여 사용해야 할 필요가 늘고 있다. 이렇게 마이크로프로세서를 내장하여 운용되는 기기를 실시간 기기(Embedded Device)라고 하며, 마이크로프로세서를 컨트롤하는 소프트웨어를 실시간 소프트웨어(Embedded Software)라고 한다[1,2].

실시간 시스템은 시스템의 동작이 그 수행완료시간에 의해서 성공 여부가 평가되는 시스템으로, 정해진 시간 안에 기능이 수행되지 못하면 시스템 기능이 정지됨으로써 그에 대한 비용을 지불해야 한다. 또한 일반적으로 시스템은 실 시간적 특징을 어느 정도는 가지고 있으며, 실시간 시스템에서는 제어 프로그램이 실시간 된 형태로 특정 기능을 제어한다. 그래서 많은 경우 실시간 시스템과 실시간 시스템은 동일한 의미로 사용되어 왔다. 미사일과 같은 군사 장비나, 통신망 제어 시스템 등이 고전적인 실시간 시스템에 해당한다.

이러한 고전적인 실시간 시스템의 경우는 그 프로그래밍의 복잡도와는 별도로 시스템 자체의 구조는 그리 복잡하지 않았으며, 실시간 소프트웨어의 프로그래밍에서는 효율성을 추구하여 주로 어셈블리 언어나 C언어, 또는 Ada와 같은 언어들이 주로 사용해 왔다[3].

그러나 제품 생산이 복잡화, 다양화됨에 따라서, 실시간 시스템은 이와 같은 고전적인 제한된 응용분야를 벗어나 인간이 개발하고 사용하는 거의 모든 기기가 실시간 시스템의 특징을 갖도록 변화되고 있다. 따라서 데스크탑 컴퓨팅 환경에서와 같은 사용자 편리성과 개발 환경의 발전을 위해서 운영체제의 필요성을 나날이 커지고 있다.

이러한 실시간 시스템은 가전 기기의 정보화와 운송기계의 기계 내부 피드백 시스템과 같은 산업용, 가정용 기기 전반에 걸쳐 확산될 것으로 전망된다. 따라서 실시간 시스템의 개발에 있어서 작업 수행의 효율성만을 추구하는 기존의 경험적이고 저 수준 프로그래밍만으로는 빠르게 변화하는 개발환경에 대응하기 어렵게 되어 가고 있다. 따라서 모듈의 재사용과 객체 지향적 소프트웨어 개발 기법, 또한 운영체제를 이용하려는 연구가 진행되고 있다[4].

2.2 목표 보드

소켓을 개발하기 위한 목표보드는 리눅스를 이식하는 실시간 시스템 보드(Embedded System Board : Target)로서 마이크로프로세서는 저 전력이며 고성능의 인텔 StronARM SA1110을 사용하였고 플래시 메모리는 인텔 플래시 16M와 32M의 SDRAM을 사용하였다. 또한 SIMM형태로 디자인되어 보드와 분리가 가능하고 모듈만을 사용하여 다른 시스템을 디자인하기가 용이하다. 또한 보드는 모니터링과 시리얼 다운로드가 가능한 시리얼 포트가 3개 있고 네트워크에 연결할 수 있도록 cs8900 이더넷 컨트롤러를 사용하여 네트워크 및 tftp 기능이 가능하다. 또한 JTAG 포트를 통하여 H/W 테스트 및 플래시에 데이터를 내려 받을 수 있다. 그리고 Character LCD와 GPIO Connector를 이용하여 디바이스 드라이버를 제작하여 테스트가 용이하도록 설계되어 있다. 커널을 포팅하기 위해서는 StrongARM CPU의 메모리 맵을 사용한다. StrongARM은 32비트 프로세서로 고정적인 메모리 맵을 가지고 있다. 전체 4Gbyte의 메모리 맵을 가지고 있으며 플래시 메모리영역이 1Gbyte까지이다. 2Gbyte까지는 내부레지스터가 연결(mapping)되어 있는 영역이고, 3Gbyte이상의 영역이 SDRAM 영역이다[5].

2.3 실시간 운영 체제로서의 리눅스

실시간 운영체제로서의 리눅스의 타당성을 고려해 보기 전에 우선 실시간 시스템의 운영체제가 갖추어야 할 조건은 다음과 같다. [8]

(1) 실시간 시스템의 운영체제는 안정적이어야 한다. 또한 실시간 시스템의 특성상 제한된 자원을 사용할 수밖에 없다. 따라서 자원의 활용이 매우 효율적이어야 한다. 즉 시스템 자원을 최대한 적게 사용하도록 커널의 크기를 작고 최적화 시켜 만들어야 하며 메모리의 관리도 효율적으로 이루어져야 한다[6].

(2) 실시간 특성을 지원해야 한다. 즉 실세계 이벤트에 대한 반응 시간을 바운드 시킬 수 있어야 한다[7].

(3) 많은 사람들이 동시에 응용의 개발에 참가할 수 있도록 공통된 표준 API를 제공해야 한다.

(4) 다양한 응용 프로그램이 지원되어야 한다.

(5) 개발의 편의를 위한 환경을 제공해야 한다.

(6) 부팅 과정이 효율적이어야 한다. 즉 필요 없는 루틴이 불필요하게 메모리를 낭비하지 말아야 한다.

2.3.1 실시간 리눅스의 특성

리눅스 소스는 웹을 통해 쉽게 구할 수 있으며, 많은 커널 프로그래머가 자유롭게 소스를 받아 버그를 수정하고 새로운 칩셋에 대한 지원을 추가한다. 따라서 새로운 장치에 대한 지원이 활발하다. 또한 리눅스는 모든 리눅스 시스템 간에 응용 프로그램이 수행될 수 있도록 표준을 제공한다. 실시간 시스템의 운영체제로 사용되는 대부분의 상용 RTOS의 경우 허가가 있는 사용자에 한해서만 응용 개발을 위한 API(Application Programming Interface)를 제공한다. 따라서 기존의 실시간 시스템에 있어서 응용 개발자는 제공되는 API를 이용해 새로운 응용을 재생산해야 하는 불편함이 있다. 응용 프로그램이 더욱 복잡해지고 커지면서 이러한 표준에 대한 중요성이 더욱 커진다. 리눅스의 장점을 요약해 보면 다음과 같다[9,10].

(1) 무료 소스(Open Source)
(2) 효율적인 시스템 관리 유틸리티들
(3) 수많은 개발 도구와 라이브러리
(4) 설계가 모듈화 되어 있고 변형이 용이
(5) 다양한 장치에 대한 지원
(6) 신뢰성 운영체제
(7) 로열티를 불필요(No licence free, no running royalty)
(8) TCP/IP 스택을 제공
(9) 세계 표준을 제공 (POSIX)

2.3.2 실시간 리눅스의 네트워크

리눅스는 네트워크 연결기능을 제공한다. 그리고 다양한 네트워크 기능들은 수많은 사용자에 의해서 검증되고 있다. 네트워크 장비의 대부분의 응용 소프트웨어는 TCP/IP 스택을 사용한다. 따라서 리눅스에서 사용되는 대부분의 네트워크 관련 소프트웨어는 네트워크 장비에 바로 사용할 수 있다[11].

새로운 네트워크 기능을 네트워크 장비에 추가할 때 실시간 시스템 전용 소프트웨어를 이용하여 개발할 경우 먼저 고가의 프로그램 소스를 사거나 직접 작성해야 한다. 그리고 컴파일한 후 실시간 시스템에 다운로드한 후 수행하고 문제가 있으면 다시 소스를 고치고 컴파일하는 과정을 반복해야 한다. 만약 동작을 잘 하지 않는 이유가 프로그램이 아니라 하드웨어 문제라면 디버깅하는데 많은 시간이 소요될 것이다. 그러나 리눅스를 사용할 경우에는 먼저 안정된 PC에서 시험할 수 있기 때문에 프로그램의 오류 여부를 쉽게 파악할 수 있다. TCP/IP 스택은 실시간 시스템과 PC 모두 같기 때문에 PC 위에서 모든 시험을 한 후 이상이 없으면 바로 실시간 시스템에 적용할 수 있다. 이로 인해 하드웨어 문제로 인한 개발 지연은 방지할 수 있다. 또한 일반 PC에서 프로그램을 개발할 수 있기 때문에 개발 인력 수급에 대한 문제도 쉽게 해결할 수 있다.

리눅스는 실시간 운영체제가 요구하는 조건을 상당부분 수용하고 있다. 또한 리눅스 네트워크 기능의 장점을 십분 활용한다면 기존의 RTOS 보다 더욱 효율적이고 경제적으로 시스템을 개발할 수 있다.

2.4 실시간 리눅스 이식

리눅스 이식이란 CPU 차원의 이식과 보드 차원의 이식으로 나눌 수 있다. CPU 차원의 이식은 크로스 개발환경에서부터 리눅스 커널에 있는 Arch 부분을 특정 CPU 환경에 맞도록 전면적으로 수정하는 것이다. 이것은 VxWork에서 사용되는 BSP에 해당된다. 즉 일체의 H/W에 관련된 리눅스 커널을 다시 만드는 작업이라 할 수 있다.

보드 차원의 포팅은 해당 CPU에 관계되는 Arch 부분이 일반 리눅스에 포함되어있거나 패치가 존재하는 경우이다. 이러한 경우에는 크로스 개발환경을 구축하고 해당 보드의 특성과 관련된 부분만 수정하여 이식하면 된다.

2.4.1 개발 환경

실시간 시스템의 운영체제를 개발하는데 제공해야할 중요한 조건 중에 하나가 개발 환경이다. 현재 몇몇 소프트웨어 회사에서 이러한 개발 환경을 제공하고 있지만 아직 실시간 시스템

전용 소프트웨어 수준의 환경은 아니다.

실시간 시스템 개발 환경은 개발 시간과 밀접한 관계가 있다. 좋은 개발 환경이라는 것은 더 빨리 개발을 할 수 있다는 것을 의미한다. 다른 분야의 실시간 시스템 개발 경우도 마찬가지지만 네트워크 장비 개발의 경우 개발 시간 단축이라는 관점에서 실시간 리눅스는 실시간 전용 소프트웨어가 가지고 있지 않은 많은 것을 가지고 있다.

실시간 리눅스를 이식하기 위한 크로스 컴파일(cross compile) 개발 환경이란 호스트 시스템(host system)에 목표 디바이스(target device)용 리눅스를 포팅(porting)하기 위한 환경을 말한다. 먼저 해당 프로세서에 맞는 툴 체인(tool chain)환경을 구축해야한다. 툴 체인이란 목표의 소프트웨어 개발을 진행하기 위해 필요한 호스트 시스템의 크로스 컴파일 환경을 말한다. 툴 체인은 각종 소스 코드를 컴파일하고 모듈을 만들어 실행 바이너리(binary)를 생성하는데 필요한 각종 Utility 및 Library의 모음이다. 기본적으로 Assembler, linker, C compiler, C Library 등으로 구성되어 있으며, SrongARM 마이클 프로세서에서 사용하기 위한 ARM tool chain은 binutils-arm-2.9.5.0, gcc-arm-2.95.2, libc-dev-arm-2.1.3, cpp-arm-2.95.2, g++-arm-2.95.2, libstdc++2.10, arm-2.95.2, libstdc++2.10-dev-arm-2.95.2 등을 사용하였다.

2.4.2 부트로더(Boot Loader)

일반적으로 부트로더라 하면 x86리눅스에선 리로(Lilo)를 많이 사용한다. 리로란 리눅스 로더로써 도스나 NT, 리눅스 등 다른 OS를 선택적으로 부팅 할 수 있도록 하는 기능을 제공한다. 리로는 하드디스크의 MBR(Memory Buffer Register)에서 동작이 되는 프로그램으로 OS가 실행 할 수 있도록 점프하는 기능을 수행하는데 비해서, 본 연구에서는 BLOB(Boot Loader Object)이란 부트로더를 사용해 플래시(flash) 0블록에서 실행되도록 하였다.

먼저 커널이나 램디스크 등의 데이터를 호스트로부터 SDRAM영역으로 다운로드(download) 할 수 있으며, SDRAM에 있는 데이터를 플래시 영역으로 쓸 수도 있다. 커널(kernel)이 이미 올라가 있다면 부팅도 가능하다.

2.4.3 커널 수정

로더가 수행을 마치면 램에는 리눅스 커널이 적재된다. 이때 리눅스는 메모리를 절약하기 위하여 로더가 사용한 메모리 영역을 덮어쓰도록 구성했다. 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 부분이 이 디렉토리를 수정함을 의미한다.

3. 실시간 리눅스 디바이스 드라이버 설계

대부분의 실시간 시스템은 프로세서, 메모리 이외에도 많은 하드웨어 디바이스로 구성되어 있다. 이들 중 일부는 응용이 필요한 디바이스이며 그 외 타이머나 시리얼 포트와 같은 일반적인 하드웨어들은 시스템에서 다양한 용도로 이용한다. 일반적으로 사용되는 디바이스 중 가장 유용한 디바이스들은 프로세서와 같은 칩에 포함되며 내장 주변기기라고 부른다. 이와 반대로 프로세서 칩 밖에 위치하는 하드웨어 디바이스를 외부 주변기기라고 부른다.

디바이스 드라이버란 운영체제와 하드웨어간의 정보 전달을 목적으로 만들어진다. 즉, 운영체제의 요구를 하드웨어에 전달하는 역할을 한다. 이러한 디바이스 드라이버들은 운영체제와 밀접한 관계를 가지게 됨으로 운영체제의 내부적인 데이터 이동을 명확하게 이해하는 것을 필

요로 한다. 또한 각각의 디바이스가 시스템에 어떻게 연결되는지에 대해서도 명확히 이해해야 한다. 예를 들면 디바이스가 ISA 슬롯(slot)에 연결되는지 아니면 PCI 슬롯에 연결되는지를 알아야 한다. 이것은 나중에 디바이스를 접근(access)하게 될 때 주소를 지정하는 것과 밀접한 관련이 있기에 이것의 이해는 필수적이다. 예를 들면 자주 사용하는 네트워크 카드는 PCI 슬롯에 끼워져서 사용되며, 디바이스의 기본 주소는 PCI 명세(specification)에 따라서 정해지며, 그것을 기준으로 디바이스의 레지스터(register)들의 번지가 정해진다. Unix에서의 디바이스는 파일(file)로서 접근 가능하며, open, read, write, close, ioctl과 같은 일반적인 인터페이스(interface)를 가지고 있다. 같은 디바이스 드라이버에 의해서 제어(control)되는 모든 디바이스들은 같은 주 번호(major number)가 주어지며, 또한 같은 주 번호를 가지는 다른 디바이스들은 부 번호(minor number)로 구분이 된다. 즉, 디바이스들은 주 번호와 부 번호의 쌍(pair)으로 구분이 가능하다[12].

3.1 I/O 접근 방법

디바이스 레지스터들에 대한 접근 방법으로는 Memory-mapped I/O와 I/O mapped I/O가 있다. Memory-mapped I/O라는 것은 메모리의 일부공간을 디바이스의 레지스터 공간으로 사상(mapping)하는 것을 의미하며, I/O mapped I/O란 특정의 포트(port)를 이용해서 디바이스와 인터페이스(interface)하는 경우이다. 본 연구에서는 Memory-mapped I/O를 이용해서 디바이스의 레지스터들에 대한 접근을 하게 되며, 이 경우에 사용하게 되는 레지스터는 GPLR과 GPDR 등이 있으며, 마이크로프로세서(StrongARM)의 28개의 GPIO 각각의 핀이 입력을 받을 것인지 출력을 할 것인지를 결정하는 레지스터이다. 이러한 기능들은 마이크로프로세서에 의존적인 부분이다. 즉, 디바이스 드라이버가 사용되는 시스템 환경(platform)에 따라서 위의 기능들은 제대로 작동하지 않을 수도 있다는 것을 고려해야 한다[13].

3.2 GPIO 디바이스 드라이버 작성 및 테스트

이 절에서는 실시간 시스템에 리눅스를 이식 후에 안정적으로 시스템이 동작하는가를 실험하기 위하여 그림1과 같이 GPIO 제어를 통해 실시간 리눅스의 디바이스 드라이버를 구현하였다. 테스트 보드 상의 있는 Push button을 눌렀을 때 몇 번 button이 눌렸는지를 테스트하는 프로그램으로 간단한 GPIO의 제어를 통하여 리눅스 디바이스 드라이버의 동작 원리를 볼 수 있다. 또한 이러한 동작 원리를 통하여 향후 네트워크 디바이스나 그밖에 다양한 임베디드 응용 디바이스 드라이버를 제작하는 기본적인 방법을 제시하였다. 본 연구에서 구현한 GPIO 디바이스 드라이버는 문자 디바이스 드라이버를 통해 인터넷과 연동하였다. 리눅스에서 문자 디바이스는 일종의 파일 오퍼레이션으로 처리한다. 먼저 GPIO 디바이스 구조를 나타내기 위한 자료구조를 정의하고 디바이스를 구분하기 위한 주번호(major number)를 결정한 후 파일 오퍼레이션을 정의하였다. 그리고 디바이스 드라이버 초기화 함수를 작성하여 드라이버 등록과 자료 구조 초기화를 한다.

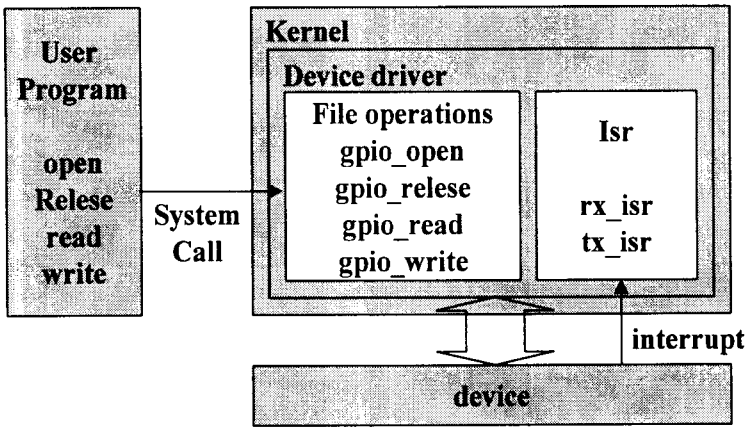

그림 1 GPIO 디바이스 드라이버 인터페이스

디바이스 드라이버는 gpio.c라는 파일이고 이 디바이스 드라이버를 테스트 할 수 있는 어플리케이션 프로그램은 test.c이다. 먼저 컴파일하면 gpio.o와 test 두 개의 파일이 생성된다. 그림 2와 같이 두 개의 파일을 목표 보드에 다운로드하고 목표보드에서 디바이스 드라이버를 적재하고 리눅스 파일 시스템에서 파일 오퍼레이션으로 처리하기 위해 i-node에 포함 시켜준 후에 테스트 프로그램을 통해 안정적으로 동작하는지 실험하였다.

GPIO는 특정 레지스터에 값을 써 주면 바로 그 값이 외부 소자에 전달되고 어느 특정 레지스터를 읽어 오면 외부에 있는 소자로부터의 값을 읽어 들일 수 있는 외부와의 통로 역할을 한다. 실험에 사용된 레지스터는 GPLR과 GPDR이다. GPDR은 StrongARM에 28개의 GPIO 각각의 핀이 입력을 받을 것인지 출력을 할 것인지를 결정하는 레지스터이다. 이것은 외부의 값을 읽어올 때와 같은 경우에 읽어 들인 값이 이 레지스터에 세팅되므로 이 레지스터를 읽어 오면 외부로부터 무슨 값이 들어 왔는지 알 수 있다.

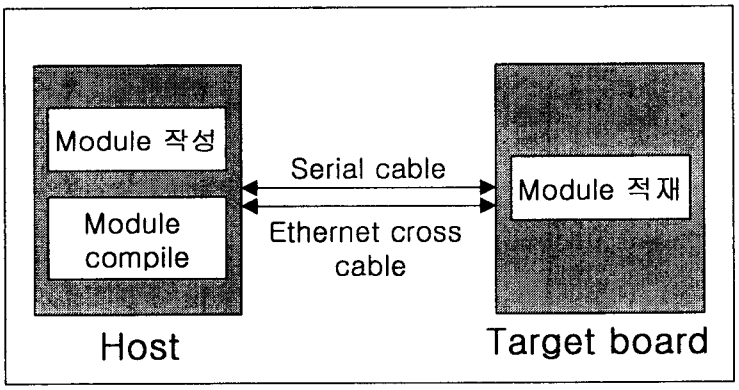

그림 2 GPIO 디바이스 드라이버 실험 환경

목표 보드로 가져온 test 파일의 속성을 실행 가능한 파일로 바꾸고, test를 실행하기 전에 GPIO 디바이스 드라이버가 제대로 적재되었는지 확인한다. 그림 3과 같이 test를 실행하여 보드상의 스위치 버튼을 누른 경우 디바이스 드라이버는 누른 핀이 high 상태이면 해당 핀의 bit

값과 누른 버튼명을 출력한다. 누른 스위치와 출력된 버튼명이 동일한지 확인하여 이상 없으면 test프로그램과 디바이스 드라이버가 올바르게 동작되는지를 알 수 있다.

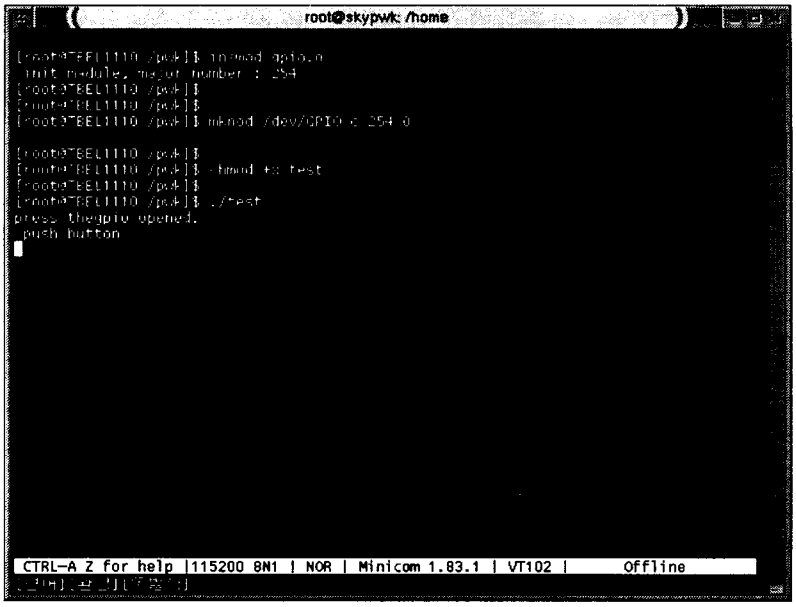

그림 3 GPIO 디바이스 드라이버 동작 실험

그림 4를 통해 GPIO button 1번에서 8번 사이에 있는 button을 눌렀을 때 메시지가 나오는 것을 확인 할 수 있다. 즉, GPIO 디바이스 드라이버가 올바르게 실행되고 있다.

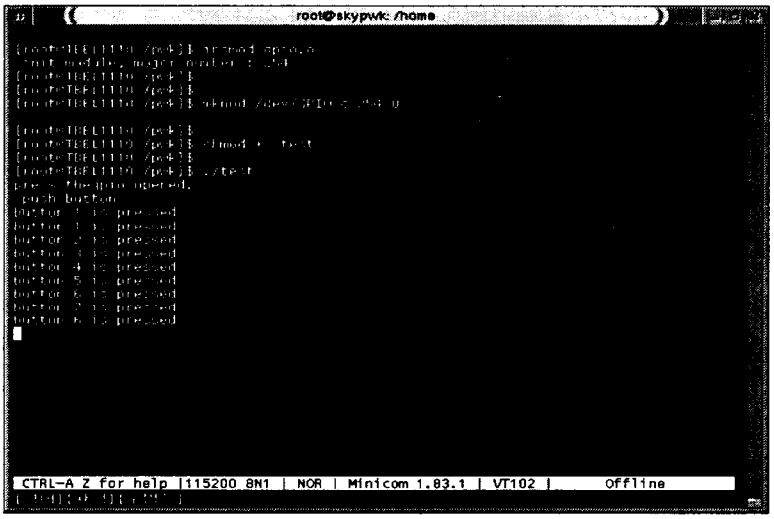

그림 4 GPIO 디바이스 드라이버 실험 결과

4. 결 론

본 논문에서는 실시간 시스템의 운영체제가 갖추어야 할 조건들을 분석하고 이러한 조건들을 기준으로 상용 RTOS들을 비교하였다. 또한 하드웨어에 리눅스를 이식하기 위한 방법을 제시하였다. 리눅스의 부트 절차의 이해와 수정해야 할 부분을 나누었다. 물론 이식하고자 할 목표 하드웨어에 따라 약간씩 차이는 있지만 운영 체제의 이식을 크게 개발 환경의 구축, 커널 로더의 개발, 커널의 수정, 그리고 응용 개발을 위한 환경 설정의 과정으로 나누어 이식을 진행하였다. 하드웨어에 다양한 설정을 디바이스 드라이버를 제작하여 실험을 수행하였으며 이를 통해 다양한 디바이스 드라이버를 제작하여 활용할 수 있다는 가능성을 보였다.

향후 Real-Time Linux는 실시간 시스템 운영체제에 많이 사용될 것으로 예상된다. 국내외에서 많은 연구가 진행되고 있으며 활성화를 위해서는 표준화와 개발 도구의 개발이 중요한 과제라고 생각한다.

5. 참고문헌

1) 이광윤, 정병수, "Embedded System의 발전 현황", 삼성 소프트웨어 센터, 2000.

2) Michael Barr, "Programming Embedded Systems in C and C++", O'REILLY, 1999.

3) Y.C. Yang and K.J.Lin. "Enhancing the real-time capability of the Linux Kernel", RTCSA'98, Hiroshima, Japan, OCT 1998.

4) Bill O. Gallmeister, "POSIX.4:Programming for the Real World", O'REILLY & Associates.

5) Intel, "StrongARM SA-1110 Microprocessor Developer's Manual", 2001.

6) Rick Grehan, Robert Moote, "Real-Time Programming : A guide to 32-bit embedded development", Addison-Wesley, 1998.

7) Peterson, silberschatz, "Operating System Conepts", Prentice-Hall.

8) Jean J. Labrosse, "μC/OS The Real-Time Kernel", R&D Publications.

9) David A Rusling, "The Linux Kernel", http://linux.flyduck.com/tlk/, 1999.

10) 임근수, "The Perfect Analysis of Linux Kernel for Implementing General Purpose Operating System", 2001.

11) 김성훈, "Linux Device Driver 설계 기술", IDEC, 2001.

12) Alessandro Rubini, "Linux Device Drivers", O'REILLY, 1998.

13) 김도형, 지현묵, 안상헌, 김준태, 조선영, "리눅스를 이용한 임베디드 시스템 구현", 비트프로젝트 57호, 2001.