ETRI

전자통신동향분석 제21권 제1호 2006년 2월

임베디드 운영체제 커널 기술 동향

Trend of Embedded Operating System Kernel Technology

임베디드 S/W 기술 동향 및
연구 개발 현황

이형석 (H.S. Lee) 임베디드OS연구팀 선임연구원
정영준 (Y.J. Jung) 임베디드OS연구팀 연구원

목 차

임베디드 시스템이라 함은 우리가 생활하는 주변에서 흔히 접할 수 있고, 얘기만 들어도 쉽게 이해할 수 있는 PDA, 스마트폰, MP3 플레이어, 셋톱박스 등의 정보가전 기기들을 비롯하여 로봇, 텔레매틱스, 공장자동화, 군사기기와 센서노드에 이르는 디지털 기기들에 컴퓨터가 내장되어 들어가 있는 시스템을 의미한다. 이러한 임베디드 시스템에 사용되는 임베디드 운영체제는 VxWorks, VRTX, pSOS, QNX와 같은 전용 RTOS에만 의존해 왔으나, 최근의 발전된 하드웨어와 네트워크 인프라로 인하여 보다 낮은 비용으로 쉽고 빠르게 개발하며 범용으로 사용될 수 있는 임베디드 리눅스를 활용하는 사례가 늘어나고 있는 추세에 있다. 이렇게 각광 받기 시작하는 임베디드 리눅스의 커널 기술과 관련한 대표적 관련 기술에 대해 설명한다.

Ⅰ. 개요

임베디드 시스템이라 함은 특정한 목적을 수행하기 위해 설계된 시스템이며, 임베디드 소프트웨어 기술이라 함은 이러한 임베디드 시스템에 내장되어 있는 소프트웨어 기술을 말한다. 데스크톱에 익숙해 있는 일반 사용자들에게는 약간 낯설 수 있으나, 실제로 임베디드 시스템은 엘리베이터, TV, MP3 플레이어, 셋톱박스, 디지털 카메라, PDA, 휴대폰, 자동차 엔진 제어, 의료기기 등 일상 생활과 매우 밀접한 관계를 갖고 있으며, 그 응용의 범위가 매우 넓다. 그래서 각 응용 임베디드 소프트웨어 시장은 응용 시장 요구가 매우 빠르게 변화하고 있다.

최근 몇 년간 임베디드 소프트웨어 기술 중 임베디드 운영체제 기술 변화의 큰 기조 중 하나는 기존 정보가전 시장과 여타의 임베디드 응용 산업을 주도하던 전통 임베디드 RTOS들인 VxWorks, pSOS, QNX, VRTX 등에 비해서 임베디드 리눅스가 크게 약진한 것이다. 그 이유는 각종 임베디드 응용에 사용되는 하드웨어의 양적, 질적 발전으로 인해 사용할 수 있는 하드웨어 자원이 늘어남에 따라, 임베디드 응용의 데스크톱화를 꼽을 수 있다. 이는 임베디드 리눅스가 기존 전통 임베디드 RTOS들보다 많은 기능을 지원하고, 공개 소스를 이용해 개발 속도가 빠르며 개발 시에 전통적인 RTOS를 사용하는 것보다 비용이 적게 드는 것에 기인한 것으로 볼 수 있을 것이다. 이처럼 최근 임베디드 리눅스가 임베디드 응용 시장에서 약진함에 따라 임베디드 운영체제 기술 동향을 알아보기 위해서는 임베디드 리눅스에 대한 연구가 필수적이므로 본문에서는 임베디드 리눅스에 대한 내용 중에서 중요한 아래의 몇 가지 기술 동향에 대해서 설명을 할 것이다.

- 실시간 지원
- 빠른 부팅 지원
- 전력 관리 지원
- 파일시스템 지원

Ⅱ. 임베디드 운영체제 커널 기술의 시장 동향

IDC의 2001년 보고서에 따르면, 네트워크 기반 임베디드 시스템에서 높은 시장 점유율을 보였던 전통 RTOS인 VxWorks, VRTX, QNX, 가전 시장에서 강세를 보였던 pSOS와 같은 전용 RTOS는 2001년을 기점으로 점차 시장 점유율이 하락하고 있다고 한다. 그 이유는 앞에서도 잠시 언급했듯 임베디드 시스템의 사용되는 응용 범위가 넓어지게 됨에 따라 필요한 많은 기능을 충족시켜 주기 위한 임베디드 시스템의 다기능화 추세를 들 수 있을 것이다. 이것이 의미하는 것은 기존의 전통 RTOS 시장이 약간씩 퇴조하며 MS의 WinCE와 임베디드 리눅스와 같은 범용 운영체제 기반의 임베디드 운영체제 중심 산업으로 재편이 되고 있다는 것을 말한다.

먼저, MS는 WinCE 3.0을 내세워 기존의 데스크톱 사용자들에게 이미 익숙해진 윈도 인터페이스를 임베디드 환경에서도 동일하게 사용할 수 있도록 제공하는 등의 기존 시장에서의 독점을 이용하여 임베디드 시장의 입지를 강화하려 하고 있다. 이에 반해, 임베디드 리눅스는 안정성과 신뢰성이 확보된 공개 기술이라는 장점과 응용 개발의 저비용의 이점을 내세우며 시장에서 각광받고 있다. 2000년대에 들어 이미 레드햇, 몬타비스타[1], 리니오 등이 임베디드 리눅스 개발 및 기술 지원 사업에 주력하고 있으며 Sharp사는 자우루스 PDA에 임베디드 리눅스를 탑재하여 상용화하는 등 임베디드 리눅스를 이용한 임베디드 응용은 급속도로 늘어나고 있는 상황이다. 임베디드 리눅스가 가지고 있는 공개 소스와 국제 표준 지원이라는 강점으로 인해 임베디드 리눅스를 이용한 각종 미들웨어 및 응용 소프트웨어를 선택적으로 탑재하는 플랫폼 기반의 소프트웨어 산업이 발전하고 있다.

일본의 경우는 1984년 도쿄대학의 사카무라 켄 교수가 주창한 “everywhere computing”을 실현

하기 위한 운영체제 규격이며 협회 이름인 TRON을 통해 미래 IT 사회 실현에 필요한 모든 시스템 소프트웨어, 도구, 응용, 기기 및 생활환경을 구성하는 임베디드 시스템 표준화를 수행하고 있으며, 이를 이용하여 일본 내의 임베디드 운영체제의 경우 약 40%가 TRON 규격을 따르고 있다. 일본을 제외한 해외에서는 아직 TRON 규격이 각광 받고 있지는 못하나, 최근 들어 임베디드 리눅스의 세계적인 기업인 몬타비스타와 제휴하는 등 활용도를 높이기 위해 노력하고 있다[2].

Ⅲ. 임베디드 운영체제 커널 기술의 동향

임베디드 운영체제 커널 기술은 임베디드 시스템에 내장되는 임베디드 소프트웨어 기술인 커널, 라이브러리, GUI, 개발 도구(넓은 의미로 컴파일러, 디버거 등도 포함) 및 응용 중 커널과 관련한 기술들을 칭하고 있다. 기술 동향과 관련해서는 앞서 설명한 바와 같이 최근 주목 받고 있는 임베디드 리눅스와 관련한 내용을 주로 다루도록 할 것이다.

먼저 임베디드 리눅스의 개략적인 특징과 최근의 응용분야는 다음과 같다.

- 임베디드 리눅스의 개략적인 특징
  - 기술적인 뛰어남, 견고성, 보안 기술 등이 검증됨
  - 좋은 개발 환경: POSIX 호환, GNU 도구 사용
  - 광범위한 응용과 하드웨어 지원
  - 다양한 종류의 네트워킹, 파일 시스템, 프로토콜 지원
  - 개방 소스 모델: 수많은 개발자들을 통한 얻기 쉬운 소스 코드
- 임베디드 리눅스의 응용분야
  - 정보가전 분야: PDA, 웹패드, 셋톱박스, 홈서버, 백색가전 등의 일반 사용자 중심의 제품
  - 휴대 통신 분야: 스마트폰, 휴대폰과 같은 휴대 통신 장비
  - 네트워킹 분야: 라우터, 게이트웨이 장비 등

이제 임베디드 운영체제 커널의 기술적 동향 중 몇 가지에 대해 알아본다.

1. 실시간 지원 기술의 동향

최근 몇 년간 관심이 집중되고 있는 임베디드 운영체제는 그 만큼 많은 발전과 더불어 많은 변화를 거치고 있기도 하다. 그 중에서도 실시간 지원 기능은 임베디드 운영체제 기술과 연관되어 있는 업체라고 한다면 임베디드 응용의 특성상 가장 중요한 몇 가지 기능 중 하나일 것이 분명하다. 그리하여, 임베디드 소프트웨어 관련 업체들이 이를 위한 기술 개발에 몰두하고 있다.

실시간 지원 기능의 차원에서 접근하여 본 최근까지의 임베디드 운영체제의 대체적인 사용 현황을 살펴보면, 정보가전과 임베디드 응용 시장 중에서도 임베디드 응용이 환경에 밀접한 연관성을 가져 경성 실시간성이 반드시 요구되는 항공 제어, 군 장비, 방어 시스템, 원자력 발전소, 의료 기기 등의 응용에서는 기존 실시간 임베디드 운영체제인 VxWorks, Greenhill, VRTX, QNX 등이 쓰이고, 연성 실시간성이 요구되는 텔레매틱스 단말기기, 실시간 멀티미디어 서비스, 네트워크 라우터, 디지털홈 분야 등에는 다양한 기능을 지원하는 임베디드 리눅스가 사용되고 있다.

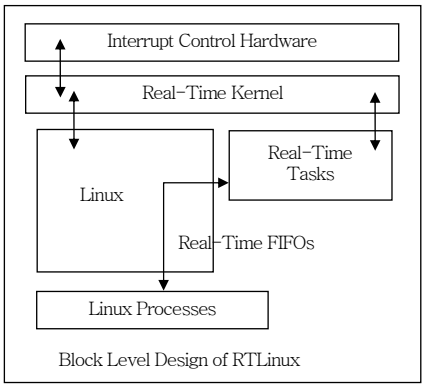
최근에는 'embedded everywhere'라는 말이 암시하듯 유비쿼터스의 열풍으로 어디서나 네트워크에 접속하여 어디에나 퍼져 있는 정보를 이용하기 위한 개인 휴대 정보단말기가 각광을 받고 있는데, 여기에는 VxWorks나 QNX 등 기존 임베디드 운영체제를 사용하는 것보다 임베디드 리눅스를 사용하는 것이 비용적인 측면과 개발의 편리성 측면에서 유리하다고 전해지고 있다. 따라서, 최근 흐름과 발맞추어 임베디드 리눅스에 관한 실시간 지원 중심으로 논한다.

가. 임베디드 리눅스 실시간 지원의 두 가지 접근 방법

임베디드 리눅스의 실시간 지원과 관련해서는 크게 두 가지 측면의 접근 방식이 있다. 첫번째 방법은 FSMLab의 RTLinux와 같이 임베디드 리눅스 커널 아래에 실시간 커널을 두어 실시간 태스크에 대한 지원을 하는 방법이고, 두번째 방법은 FSMLab를 제외한 몬타비스타와 같이 대부분이 전통적인 임베디드 리눅스 커널에 수정을 가해 실시간성을 지원하고 있는 방법이다.

1) RTLinux의 subkernel을 이용한 실시간 지원 방법

첫번째 실시간 지원 접근 방법인 RTLinux는 (그림 1)[3]에서 보듯이 하드웨어 바로 위에 FSMLab에서 개발한 실시간 커널이 존재하고 그 위에 실시간 태스크들이 독립적으로 존재하며, 리눅스 커널은 실시간 태스크가 없거나 현재 CPU를 점유할 필요가 없는 경우에 실시간 커널로부터 CPU를 할당 받아 돌아가는 가장 낮은 우선순위를 가지는 태스크라고 볼 수 있다. 실시간 커널에는 실시간 지원 스케줄러를 이용한 실시간 스케줄링을 통해, 실시간 태스크에 대한 지원을 하게 된다. 또, 실시간 태스크들과 리눅스 태스크들간의 통신을 위해서 실시간 FIFO를 제공한다[4].



(그림 1) RTLinux의 블록 다이어그램

RTLinux는 현재 항공기 제어 시스템, 군사 기기, 산업용 로봇, 의료 기기 등의 분야에 적용되며, 경성 실시간성을 지원하고 있다. 그 반면 RTLinux는 POSIX의 일부만을 지원하고 있으며, 실시간 태스크들을 위한 API를 지원하고 있어 POSIX 응용과의 호환에서는 일부 포팅이 필요할 수 있다. 또한, 이 문제는 디바이스 드라이버와 관련해서도 나타날 수 있는 문제이다. 그리고, 아직까지는 리눅스 커널과 따로 떨어져서 사용할 수는 없다. 그래서 커널 사이즈도 임베디드 리눅스 커널 + FSMLab 실시간 커널 정도의 크기라고 봐야 할 것이다. 따라서, RTLinux의 장점은 경성 실시간 시스템을 요구하는 특정 시스템에서의 포팅을 전제로 사용이 용이한 것으로 보이며, 이러한 특성에 덧붙여 리눅스 커널의 다양한 기능까지 이용할 수 있다는 것이다.

2) 표준 임베디드 리눅스 커널 구조의 수정을 통한 실시간 지원 방법

두번째 실시간 지원 방법은 몬타비스타를 위시한 여타의 임베디드 리눅스 벤더들과 오픈 커뮤니티에서 사용하는 방법으로 일반적인 임베디드 리눅스 커널에 실시간적 요소를 추가적으로 지원하는 방법이다. 이 방법은 앞서 설명했던 RTLinux와는 달리 연성 실시간성을 지원하지만, POSIX 응용과 호환을 이루어 기존에 사용하던 임베디드 리눅스 응용들을 그대로 사용할 수 있고, POSIX와 호환을 이루기 때문에 개발자들에게 응용 개발에 대한 투명성을 제공할 수 있다는 것이 가장 큰 장점이다. 이에 대한 실시간 지원 방법은 크게 선점형 커널(preemptible kernel) 지원, 락 브레이크(lock break) 기법 지원, 실시간 지원 스케줄러의 세 가지로 나눌 수 있다. 다음은 이에 대한 구체적 설명을 기술한다.

- 선점형 커널 지원

임베디드 리눅스에 있어서 실시간성 지원의 핵심 중 하나는 선점성(preemptibility)을 높이는 것이라고 말할 수 있다. 그 이유는 어떤 운영체제가 실시간성을 갖는다는 것은 어떤 한 시점에서 수행되고 있

는 일반적 태스크보다 우선순위가 높은 실시간 태스크가 발생했을 때, 제한된 시간 내에 발생된 실시간 태스크의 작업 수행 시작과 작업 수행 완료의 시간적, 작업수행 정확도의 보장이 이루어져야 됨을 의미한다. 이 중에서도 시스템의 시간적 보장을 위해서는 전체 시스템을 이루고 있는 각 부분에서 동기적, 비동기적 이벤트에 대해 얼마나 즉각적인 반응을 보일 수 있는지에 대한 응답성(responsibility), 선점성이 중요해진다.

그러나, 임베디드 리눅스의 커널은 일단 일반적 태스크가 시스템 콜(system call)을 사용하여 커널 내부로 진입하게 되면 현재 수행중인 태스크의 우선순위보다 높은 우선순위의 실시간 태스크가 발생한다 할지라도 실시간 태스크가 즉각적으로 수행될 수 없다. 그 이유는 리눅스가 전통적인 유닉스의 모노리틱(monolithic) 커널의 성격을 띠고 있어서 처리량(throughput)을 중시하는 구조이기 때문이다. 따라서, 실시간성 지원이 이루어지기 위해서는 커널 내부의 구조가 선점성이 높은 구조로 변경되어야 한다. 즉, 커널 내부의 보호받아야 할 최소한의 커널 코드를 제외한 지역에서는 선점(preemption)이 가능해야만 한다. 따라서, 이를 가능하게 하기 위해 커널의 락메커니즘(lock mechanism)을 수정하여 임베디드 리눅스 커널을 선점형 커널이 되도록 하여 실시간성을 지원하는 것이다. 임베디드 리눅스 커널의 락메커니즘 수정은 SMP의 락메커니즘으로 사용하는 스핀락(spin lock)이 커널 내부의 선점으로부터 보호받아야 할 최소한의 임계지역(critical section)과 거의 일치함을 이용하여 스핀락 구간 내에서는 선점이 일어나지 못하게 하고, 이 구간을 빠져나올 때 선점이 되어야 하는지 아닌지를 판단하여 선점이 필요한 경우 선점이 일어나도록 하고 있다[5]. 이 방법은 리눅스 커널 버전 2.4에 와서 가능하게 되었는데, 리눅스 커널 버전 2.2와 달리 버전 2.4에서는 스핀락이 더 세분화(fine grained)되어 커널 내의 짧은 스핀락 구간을 제외한 나머지 구간에서는 선점이 가능하도록 되었으며, 그것의 효율성이 있는 것이다. 또한, 리눅스 커널 2.5.4에서 포함되어 공식 안정화 리눅스 커널 버전 2.6에는 기본으로 포함되어 커널 컴파일 시에 옵션으로 선택할 수 있다. 이렇게 하기 위한 수정된 락메커니즘의 개념적인 의사코드(pseudo code)는 아래와 같다.

```
/* 선점 가능 지역 */
preempt_lock();
spin_lock();
/* 선점 불가능 지역 */
spin_unlock();
preempt_unlock();
/* 선점 가능 지역 */
```

이 기능을 사용하게 되면 커널 내부라 할지라도 스핀락이 걸려 있는 구간을 제외한 타지역에서는 선점이 가능한 선점형 커널을 구성할 수 있게 된다.

• 락 브레이크(lock break) 기법 지원

임베디드 리눅스 커널은 공개 소스 기반으로 다수의 개발자들이 참여하여 개발되어온 역사에 따라 실시간성을 높이기 위한 응답성을 중시하기 보다는 안정성과 처리량을 중시하는 것은 이미 알려진 사실이다. 이에 따라 개발자들이 개발 당시의 커널 곳곳의 코드에 안정성을 높이기 위한 긴 락 구간(long-held locked region)이 많이 있는 편이라 할 수 있다. 이러한 긴 락 구간은 커널이 선점형 커널이라 할지라도 긴 락 구간 내의 태스크들은 선점할 수가 없고 시스템의 응답성이 현저히 줄어들 수 밖에 없으므로, 이러한 코드 구간을 실험을 통하여 찾아내어 가능한 경우에 짧은 응답시간을 갖도록 락 구조를 변경하도록 한다. 이 방법은 임베디드 리눅스 커널의 응답성을 높이는 데 공헌을 많이 하고는 있으나, 이 구간을 실험적으로 찾아내기가 어렵고 찾아낸다 하더라도 때에 따라서는 락 구조를 변경할 수 없는 경우가 있어 쉽지 않은 실시간 지원 방법이기도 하다. 일반적으로 실험적으로 이러한 긴 락 구간을 찾아본 결과 비교적 가상 파일 시스템 관련 코드가 주 수정 대상이다. 다음은 수정된 의사코드의 한 예이다[6].

```
void prune_dcache(int count)
{
DEFINE_RESCHED_COUNT;
redo:
    spin_lock();
    for(;;) {
        …
        if (TEST_RESCHED_COUNT(100)) {
            RESET_RESCHED_COUNT();
            if (conditional_schedule_needed()){
                spin_unlock();
                unconditional_schedule();
                goto redo;
            }
        }
    }
    …..
    spin_unlock();
}
```

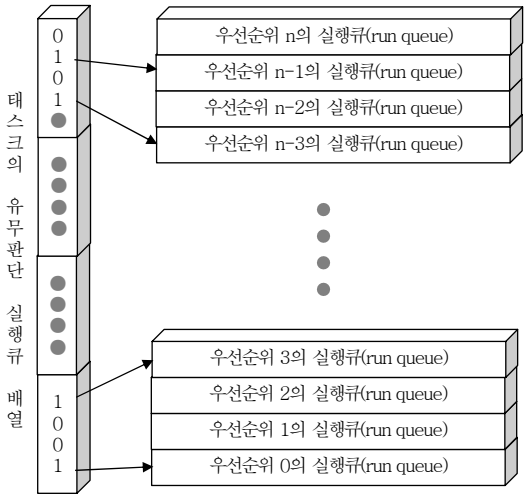
이 코드는 dcache.c라는 데이터 캐시 제어 부분으로, 사용되지 않는 데이터 블록을 디스크로 보내는 역할을 한다. 이때, 코드 내에서 데이터 블록을 다 보낼 때까지 무한루프를 수행하는데, 계속 수행하지 않고, 루프를 돌다 특정 횟수가 되면 선점될 필요가 있는지의 여부를 자체적으로 검사하여 필요한 경우 선점이 이루어지도록 하고 있다. 이렇게 커널 내부를 전체적으로 선점형으로 하지 않아도, CPU를 많이 소모하는 커널 코드를 수정해줌으로써 실질적인 빠른 응답성의 효과를 얻도록 하고 있는 것이다.

• 실시간 지원 스케줄러

임베디드 리눅스의 커널 버전 2.6 이전의 스케줄러 구조는 실시간 태스크와 일반 태스크를 분리하여 다중 실행큐(run queue)를 구성하지 않고 하나의 실행큐를 적재하여 사용했다. 이는 임베디드 리눅스 커널의 원래 취지인 처리량 지향적 방식에 기인한 것인데 이럴 경우에는 스케줄러가 호출되어 시스템 내의 태스크들에 대한 재스케줄링이 이루어질 때 루프를 돌면서 실행큐에 적재된 모든 태스크들을 검사해야 하므로, 실행큐에 적재된 태스크 수에 따라 스케줄링 시간의 변화가 생길 수 밖에 없다. 따라서, 이런 구조로는 태스크 수에 상관없이 실시간 태스크들에 대한 고정적인 스케줄링 시간을 갖는 스케줄러를 기대할 수가 없으므로, 실시간 지원적 성격의 하나인 예측가능성(predictability)이 낮아질 수밖에 없다. 이러한 문제점을 해결하기 위해 고정 스케줄링 시간 지원 스케줄러를 지원하는데 커널 2.6에서 지원되며 통상적으로 O(1) 스케줄러로 알려져 있기도 하다. 이 스케줄러에는 추가적인 기능들이 있긴 하지만 주된 기능은 태스크 수의 증감에도 고정적인 시간내 실시간 태스크의 구동을 위한 멀티 피드백 실행큐(multi feedback run queue)를 사용한다는 점이다. (그림 2)는 이에 대한 구조도이다[7].

각 우선순위마다 다른 실행큐를 가지고 있으며 해당 우선순위의 실행큐를 찾아가는 방법은 비트맵 구조를 이용하여 서로 다른 우선순위를 가지는 태스크의 수가 많아져도 제한된 시간 내의 스케줄링 지연시간을 가질 수 있도록 해준다.

이상에서 임베디드 리눅스 커널에 대한 실시간 지원에 대한 기술적인 설명을 했으며, 이제는 이에 대한 응답성 비교를 제시한다. 비교 대상은 일반적인 임베디드 리눅스 커널 버전 2.4.18과 앞서 언급했던 선점형 커널 기능, 락 브레이크 기법을 같은 임베디드 리눅스 커널 버전에 적용한 상태에서의 비교를 한다. 비교 환경은 2GHz의 CPU, 256M의 메모리를 가지는 데스크톱 환경이며, 시스템 반응 시간



(그림 2) 멀티 피드백 실행큐 구조

을 측정하기 위해 DVD 플레이어를 10시간 구동시키며 스케줄링 지연시간을 측정했다. 그 결과는 <표 1> 및 (그림 3)의 그래프에 잘 나타나 있다.

(그림 3)에서 표시한 'vanilla kernel'은 일반 임베디드 리눅스 커널을, 'preempt kernel'은 선점형 커널을, 'lockbreak kernel'은 락 브레이크 기법 지원 커널을 의미하나 이 속엔 선점형 커널의 기능을 포함하고, 'low latency kernel'은 락 브레이크 기법과 유사하나 선점형 커널의 기능을 포함하지 않고 있다. 표준형 커널은 ETRI에서 개발한 커널로 실시간 지원적인 측면에서는 실시간 지원 스케줄러와 보다 향상된 락 브레이크 기능을 지원하고 있다. <표 1>에서 '99.999%'가 의미하는 것은 10만 번 중 99999번은 실시간성을 보장하지만 한 번은 보장하지 못하는 시간을 의미하는 것으로 연성 실시간성을 의미하는 것이라 보면 된다. 또한 <표 1>에서 보듯이 임베디드 리눅스 커널의 평균 스케줄링 지연시간은 약 5μsec에 불과하지만 최악 스케줄링 지연시간은 약 169msec로써 엄청난 편차를 가지고 있다. 여기에 실시간 지원 기능을 추가하여 많은 응답성 향상을 가져왔음을 알 수 있다.

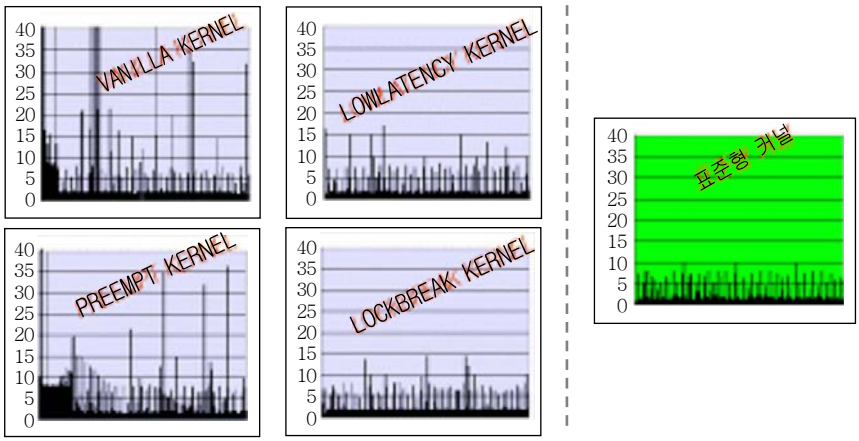
〈표 1〉 스케줄링 지연시간 비교표

커널 \ 사양	평균 스케줄링 지연시간 (μsec)	최대 스케줄링 지연시간 (msec)	99.999% 이내(msec)
Vanilla	5.719	169.050	17.84
Preempt	4.602	53.809	7.35
Lockbreak	4.305	14.575	1.49
Lowlatency	4.125	16.157	1.49
표준형 커널	3.751	9.544	1.47

나. 임베디드 리눅스의 경성 실시간 지원 요구

2000년 이전까지 기존의 전통적인 RTOS가 주도하던 임베디드 시스템 시장이었고, 그 이후 최근까지가 기존의 전통적인 경성 실시간 지원 RTOS의 퇴조와 임베디드 리눅스의 약진이라면, 요 근래의 분위기는 임베디드 리눅스 시장의 강세와 임베디드 리눅스의 경성 실시간 지원성이 핫이슈로 부각되고 있다. 특히, TimeSys, 몬타비스타 및 NEC, SONY, IBM, 미쯔비시, 삼성, LG 등의 주로 가전회사들과 임베디드 리눅스 회사들에 의해 2003년 7월 결성된 CELF[8] 등에서 임베디드 리눅스에 경성 실시간성을 지원하기 위해 노력하고 있다. 그리하여, 수백 KByte 초반에서 100KByte 이하의 각종 임베디드 시스템에 탑재하여 그 활용성을 최대한으로 높이기 위함이다.

임베디드 리눅스의 태생적 한계로 인한 경성 실시간 지원의 어려움에도 불구하고 이들이 경성 실시간성을 지원하고자 하는 이유는 기존 RTOS 시장에서 각광 받던 운영체제들보다 비용면에서 많은 절감



(그림 3) 스케줄링 지연시간 비교 그림

을 가져올 수 있을 것으로 예측되기 때문이다.

국내에서도 ETRI에서 임베디드 소프트웨어 표준 플랫폼 사업의 일환으로 산업체의 임베디드 시스템에서 규모별(① 400~500KByte 이상, ② 100~200KByte, ③ 수십 KByte. 단 커널 크기 기준)로 활용할 수 있도록 연구를 하고 있으며, 국내외적인 추세로 비추어 볼 때 반드시 필요한 연구로 보인다. 이미 VxWorks로 유명한 WindRiver도 리눅스를 지원하면서 이 작업을 시작했으며, 일본에서도 TRON 프로젝트의 한 세부 프로젝트인 T-Engine 프로젝트에서 일본의 산업계에 지원하기 위한 규모별 운영체제 사업을 이미 시작했기 때문이다. 이런 산업계의 임베디드 리눅스의 경성 실시간 지원성을 갖추게 될 경우 기존 전통 RTOS의 값비싼 로열티를 물지 않고 대체하는 효과를 낳을 수 있을 것이다.

2. 빠른 부팅 지원

x86 구조를 이용하는 임베디드 시스템에 있어서 빠른 부팅은 주요한 이슈 중 하나이다. x86 구조에서 부팅을 하기 위해서는 바이오스의 초기화를 거쳐야 하는데, 보통 바이오스의 초기화에만 10~20초 이상의 많은 시간이 소요되어 홈서버나 정보가전기기용으로 사용하기에는 너무 느리다는 문제점이 있었다. 이에 대한 지원 방법으로는 공개소스 BIOS 프로젝트인 LinuxBIOS[9]가 있으며, 다음은 그에 대한 개략적인 설명이다.

LinuxBIOS는 Los Alamos National Laboratory의 Ron Minch 등에 의해 기존의 BIOS를 대체하려는 목적으로 시작된 공개소스 프로젝트이다. LinuxBIOS의 가장 큰 특징은 로열티가 없고 매우 빠르며 대부분의 코드가 C로 작성되어 있다는 점이 특징이다. 특히 부팅 시간에 있어서 통상의 BIOS에서 시스템 초기화에 약 10~20초의 시간이 소요되는 데 반해 LinuxBIOS에서는 1초 이내에 초기화를 완료하고 운영체제를 로딩하는 것이 가능하다. 이러한 차이는 기존의 바이오스가 16bit DOS 운영체제와의 호환성을 위한 불필요한 오버헤드가 크기 때문이다. LinuxBIOS에서는 불필요한 오버헤드를 최소화 함으로써 빠른 부팅을 가능하게 한 것이다. 구체적으로는 부팅 직후 32bit 프로텍티드 모드로 전환하며, 메모리 및 캐시의 초기화를 최우선으로 하고, 최소한의 하드웨어 초기화만을 수행하고 나머지는 운영체제에게 맡기는 방식으로 시간을 최소화하고 있다. 일반적인 LinuxBIOS의 수행 순서는 다음과 같으며 이 모든 과정이 1초 이내에 수행된다.

Step 1) PC reset, jump to 0xffff:0000
Step 2) Turn on GDT in crt0.S
Enter 32bit Protected mode
Step 3) Turn on minimal serial support if needed
Step 4) Turn on RAM (SDRAM or DDR SDRAM)
Initialize and check memories
Initialize stack and heap
Jump to C
Step 5) Enable Cache and remaining setup
Configure MTRR and enable L2 Cache
_____PCI initialize
Step 6) Loading Linux (or bootloaders)

이와 같이 LinuxBIOS를 사용함으로써 비용의 절감과 빠른 부팅이 가능하다는 장점이 있지만 현재로서는 지원되는 칩셋이 매우 제한적이며 VGABIOS와의 호환 문제 등 여러 가지 문제점 또한 존재한다. 이중 가장 큰 문제는 제한된 칩셋 지원인데, 여러 개발자들에 의해 점차적으로 지원되는 칩셋이 늘어나고는 있으나 그 수는 아직 제한적이다.

ETRI에서는 지난 2년에 걸쳐 LinuxBIOS 프로젝트에 참여하면서 초기 버전의 홈서버에서 사용하였던 인텔 I815EP 칩셋과 현재의 홈서버에서 사용하는 SIS 645, 961 칩셋에 최초로 LinuxBIOS를 포팅했다.

새로운 칩셋을 지원하기 위해 해야 할 일 중 가장 중요한 일이자 가장 어려운 일은 DRAM 초기화 루틴의 작성이다. 앞서 언급했듯이 LinuxBIOS의 주요 코드는 C로 작성되어 있기 때문에 반드시 스택이 존재해야 하며 이는 DRAM의 동작이 필수적이다. 때문에 DRAM 초기화 코드는 어셈블리로 작성되며 LinuxBIOS의 가장 초기에 수행된다. DRAM 초기화 방법은 칩셋마다 다르기 때문에 칩셋 제조회사의

데이터 시트에 의존해서 작성해야만 한다.

DRAM의 초기화 루틴의 작성이 끝나면 C로 작성된 LinuxBIOS의 코드를 수행할 수 있으며 C언어로 기타 칩셋의 초기화 루틴(내장 USB 컨트롤러 초기화 등)을 작성한다. 이러한 과정이 끝나면 Linux를 부팅할 수 있다. BIOS로 할당된 내장 flash의 크기가 512KB면 LinuxBIOS와 함께 Linux 커널 또한 flash 안에 넣는 것이 가능하며 256KB flash인 경우 Etherboot나 LILO 등의 bootloader 프로그램을 통해 Linux를 로딩할 수 있다.

3. 전력관리 지원

가. 전력관리 개요

전력은 시스템의 하나의 자원이지만 전원이 연결되어 있는 시스템에서는 거의 무한한 자원이라고 볼 수 있기 때문에 최근까지 운영체제에서의 전력관리에 대한 고려가 없었으나 이동 기기의 증가와 이에 비하여 배터리의 느린 발전 속도로 인하여 중요한 자원으로 인식되어 지고 있으며 운영체제의 성능을 가름하는 주요 요인으로 자리를 잡게 되었다. 전력관리는 기본적으로 시스템의 성능의 감소를 최소화하면서 시스템이나 기기가 사용되지 않을 때 전력상태를 저전력 상태로 천이시킴으로써 불필요한 전력소모의 낭비를 막는 것과 시스템이나 기기의 작업의 변화를 통해 저전력 상태로 존재하는 시간을 최대화하는 것을 목적으로 한다. 이를 위해 시스템의 workload 상태에 따라서 동적으로 전력 상태를 변환시키는 기술이 필요한데 이를 동적 전력 관리라고 하며 ETRI에서는 효율적인 동적 전력 관리에 대한 연구가 이루어지고 있다.

나. 전력관리 기술 동향

동적 전력 관리는 하드웨어 리소스의 전력 상태를 동적으로 모니터링 및 관리함으로써 전력소모를 최소화하기 위한 것이다. 전력관리는 크게 하드웨어 인터페이스 부분과 OS에서 하드웨어 전력 상태 변환을 담당하는 PM engine 및 시스템을 모니터링하여 적합한 전력 상태를 결정하는 policy manager로 나뉠 수 있다.

OS와 하드웨어의 인터페이스는 x86 계열을 중심으로 ACPI를 통해 리눅스 상에 구현이 되어 있으며 이를 이용하여 PM engine, policy manager의 연구 및 구현이 현재 활발하게 진행되고 있다. 2002년 말에 IBM과 MontaVista는 리눅스 상에서 DPM이라는 PM engine을 PowerPC 405LP에 구현하여 발표하였고 현재는 이 DPM을 기반으로 policy manager를 추가한 specification 및 참조 구현을 CELF에서 진행하고 있다.

학계에서는 UC Irvine의 OSDPM project를 통해서 Dynamic Power Management Framework를 위한 연구를 진행중이며, 이 외에도 많은 대학에서 DPM 및 power aware computing에 관련된 많은 연구가 이루어지고 있다.

이와 관련하여 ETRI에서는 애플리케이션의 전력 및 성능 요구사항을 반영하고 시스템의 전력 상태를 모니터링하여 전력 효율을 극대화하기 위한 '능동형 전력관리를 위한 Policy Manager'를 설계 및 구현하는 연구를 수행하고 있으며 세부 사항은 아래와 같다.

- 애플리케이션에서의 전력 및 성능 요구를 반영하여 전력관리를 수행하는 전력관리 policy manager
- 전력관리를 policy라는 디바이스들의 전력 상태의 집합 형태로 관리하기 위한 프레임워크 개발
- 애플리케이션의 전력 및 성능 요구사항에 맞춰주기 위한 시스템의 전력 상태 모니터링 모듈(특히 동적 전압 변동 관련한 커널 모니터링을 위한 기능 구현)
- 참조 애플리케이션(예를 들면 MPEG player) 구현을 통한 성능 평가 및 테스트 수행

4. 파일시스템 지원

홈서버, 셋톱박스, 텔레매틱스 단말 등의 임베디드 시스템은 가격 경쟁을 위해 일반 PC보다 저사양

의 성능을 갖는 시스템으로 구성되는 것이 보통이지만, 멀티미디어 데이터 처리에 대해 과도한 입출력 전송을 요구하는 것이 보통이다. 기존의 텍스트나 이미지 데이터와 달리 멀티미디어 데이터는 실시간으로 일정 시간까지 요청한 데이터를 서비스해 주지 못할 경우 사용자 측면에서 화면의 일그러짐, 끊김 등의 현상이 발생하여 원래 자료의 의미가 제대로 전달되지 못하게 된다. 따라서, 주어진 데이터를 처리하는 소프트웨어 시스템이 매우 효율적으로 설계되어 있어야 이러한 현상을 최소화 할 수 있다. 이를 위하여 멀티미디어 지원 파일시스템이 절실히 필요하다. 예전에는 VOD 서비스를 위한 서버에 이러한 파일시스템을 개발하였으나, 최근에 임베디드 시스템의 요구에 의해 개발되고 있다[10],[11].

기존의 유닉스 계열 파일시스템은 데이터 블록 단위로 파일을 배치하기 때문에 멀티미디어 데이터의 대용량 파일에 대해서는 디스크 단편화가 발생할 수 있다. 즉, 멀티레벨 i-node 구조는 응용 프로그램의 순차적 읽기 명령에서 디스크 헤드를 과도하게 움직여 많은 오버헤드를 유발하게 된다. 최근에 개선된 각종 파일시스템 기법(블록 그룹, 실린더 그룹 등)이 이러한 디스크 헤드의 움직임을 줄이긴 했으나, 아직 근본적인 문제가 존재한다. 이를 개선하기 위한 멀티미디어 파일시스템은 멀티레벨 i-node 구조를 피하고, 데이터 저장의 최소 단위를 Extent로 변경한다. 여기서, Extent는 여러 개의 데이터 블록 크기로 구성되는 멀티미디어 파일의 가장 기본적인 구성 단위이다. 또한, 멀티미디어의 자료 중간에 디렉토리의 내용이 들어가는 것을 막기 위해, 디렉토리 내용은 디렉토리 Extent에 따로 저장한다. 이렇게 분리해서 저장함으로써 파일 종류에 따른 지역성(locality)을 높여 디스크 탐색(seek) 시간을 줄일 수 있다.

멀티미디어 스트림의 QoS를 보장하기 위해서는 디스크 저장 구조만을 변경하는 것으로는 부족하다. 멀티미디어 데이터에 적합한 스케줄링 방법이 필요하다. 현재의 리눅스 시스템에서 디스크 스케줄링 기법(엘리베이터 알고리듬)은 디스크 헤드의 이동 시간을 최소화하는 데 중점을 두고 있고, 입출력 요청들의 다양한 요구조건은 고려하지 않고 있다. 따라서, 멀티미디어 재생중에 FTP 등의 다른 서비스를 이용하게 되면 화면의 끊김 등이 발생하여 데이터를 제대로 처리할 수 없다. 이러한 현상은 응용 프로그램마다 다른 입출력 특성을 요구하는 데 반해 하위 레벨에서는 이를 구분할 수 없기 때문이다. 이를 해결하기 위해 실시간 스케줄링 방법이 필요하다. 리눅스의 엘리베이터 알고리듬 대신에 SCAN-EDF를 사용한다. SCAN-EDF는 데드라인이 가장 급한 순서대로 스케줄링하는 EDF를 먼저 사용하고 데드라인이 같을 경우 SCAN 방식을 사용함으로써 디스크 입출력 처리율을 높인 방법이다. 또한, 멀티미디어 서비스에 대한 여러 요청이 들어올 경우를 관리하기 위해 수용 제어모듈(admission controller)을 도입하여 멀티미디어 서비스를 관리할 수 있다.

이러한 임베디드 시스템 환경의 파일시스템은 임베디드 시스템의 특성으로 인해 갑작스런 전원 차단의 문제에 대한 대비책이 필요하다. 전원차단으로 인한 데이터의 손실이 없어야 하며, 무결성을 보장해야 한다. 이를 위해 기존의 파일시스템에서는 다시 부팅이 이루어진 뒤 파일 시스템 일관성 체크, 즉 fsck(filesystem consistency check)를 수행하여 이를 관리했다. 그러나, 이런 방식은 시스템에 따라 일반적으로 수 분에서 길게는 수 시간까지 걸리기 때문에 임베디드 시스템에서 사용할 수 없으며, 새로운 방식이 필요하게 되었다. 이런 불합리한 문제를 해결하고자 하는 것이 저널링 파일시스템이다. 저널링 파일시스템은 저널(journal)이라고 하는 새로운 데이터 구조를 추가함으로써 fsck 문제를 해결한다. 파일 시스템이 메타 데이터에 어떠한 변경을 가하기 이전에 어떤 일을 할 것인지에 대한 내용을 저널에 기록한다. 그 다음 메타 데이터를 수정하게 된다. 그렇게 함으로써 저널링 파일시스템은 최근의 메타 데이터 수정사항의 로그를 유지하며, 이는 비정상적인 시스템 종료에 따른 파일시스템의 무결성 보장에 유용하게 사용된다.

최초에 저널링 파일시스템은 시스템 가동시간이

중요시 되는 데이터 센터환경에서 엄청난 시간을 요구하는 fsck를 줄여보고자 하는 노력에서 도입되었다. 하지만 최근에 일반 PC 환경인 리눅스의 ext3, ReiserFS, XFS, JFS 등과 윈도 XP의 NTFS 등에서도 저널링 기능을 탑재하고 있으며, 임베디드 환경의 파일시스템으로 확장되고 있다. 몇 가지 저널링 파일시스템들의 특징에 대하여 알아보면 다음과 같다.

XFS[12]는 SGI사가 1994년 12월에 IRIX 5.3 OS에서 개발하여 IRIX 6.2부터는 SGI가 판매하는 모든 시스템에 기본적으로 제공한 저널링 파일시스템이다. 1999년 초부터 SGI에서 이 파일시스템을 리눅스에 포팅하기 시작했다. XFS는 64비트를 지원하는 파일 시스템으로 시스템에서 사용되는 각 파일의 disk block offset이나 i-node number assign 등이 모두 64비트를 제공한다. 이 파일시스템은 AG에 의해 여러 개의 영역으로 나뉘어 관리된다. 그리고 자유 블록 및 아이노드 맵을 B+ 트리 형태로 관리하여 검색 및 할당을 일반적인 순차 검색보다 빨리 할 수 있다.

JFS[13]는 IBM사가 판매하는 엔터프라이즈 서버 등에 탑재하여 제공하던 로그기반 파일 시스템이다. IBM에서는 이 JFS를 1999년부터 리눅스에 포팅하기 시작하였다. 메타 데이터만을 로깅하며 성능상의 이유로 비동기적인 로깅을 지원한다. Extent 기반의 디스크 블록을 할당하며 XFS에서처럼 블록 및 아이노드 맵을 B+ 트리를 사용하여 관리한다. 그리고 파일 시스템의 I/O 단위로는 512, 1024, 2048 및 4096 바이트 등 다양한 블록 사이즈를 지원한다.

ReiserFS는 NameSys에서 리눅스 2.2.14를 기반으로 개발한 파일 시스템으로 메타 데이터만을 로깅하며 작은 파일 여러 개를 하나로 묶어 관리한다. 자원 할당 맵으로 B* 트리를 사용하여 관리하고 이름 공간 할당에 의해서 발생될 수 있는 단편화 요소를 줄였다.

Stephen Tweedie 박사가 설계한 ext3 파일 시스템은 기존의 ext2 파일 시스템의 프레임워크에 내장된다. ext3은 저널링을 지원한다는 점을 제외하고는 ext2와 매우 유사하다. 다른 저널링 파일 시스템처럼 B+ 트리를 사용해 자원의 할당을 돕는다든지 i-node에 데이터를 저장하는 등의 노력을 하지 않는다. ext2의 구조를 유지하기 위해 ext3은 저널링 부분을 전담하는 JBD라는 일종의 블록 디바이스 드라이버를 도입하였다. 이것은 파일 시스템이

〈표 2〉 저널링 파일 시스템 비교

	ReiserFS	XFS	JFS	ext3
최대 파일시스템 크기	16TB	18 thousand PB	4PB~32PB	4TB
최대 파일 크기	4GB	9 thousand	512TB~4PB	2GB
Block 크기	Up to 64KB	512B~64KB	512, 1K, 2K, 4K	1K~4K
Free block 관리	Bitmap	B+ Tree	Binary Buddy	
File block 관리	B* Tree	B+ Tree	B+ Tree	
Directory 관리	B* Tree	B+ Tree	B+ Tree	
Free space의 Extent 관리	No	Yes	No	
File block의 Extent 관리	No	Yes	Yes	
I-node에 symbolic link 저장	Yes	Yes	Yes	same as ext2
I-node에 데이터 저장	Yes	Yes	No	
동적 i-node 할당	Yes	Yes	Yes	
동적 i-node 관리	B* Tree	B+ Tree	B+ Tree	
Sparse file 지원	Yes	Yes	Yes	
ACL 지원	No	Yes	Yes	

기록하기를 원하는 데이터 블록을 전달 받아 자신의 블록 장치에다가 주기적으로 쓰는 작업을 하게 된다. 또 필요한 경우 기록한 데이터를 복구하기도 한다. 특히 JBD는 파일 시스템에 종속적으로 설계되지 않았기 때문에 저널링을 탑재하고자 하는 어떤 블록장치 기반 파일 시스템에서도 사용할 수 있다.

<표 2>는 이러한 저널링 파일 시스템의 특징을 비교한 표이다.

기존 임베디드 리눅스 제품의 파일시스템은 리눅스에 포함되어 있는 자체의 파일시스템을 사용하고 있으며, 그 외에 특별히 임베디드 시스템에 맞추어 개발된 파일시스템은 거의 없으며, 홈서버나 인터넷 TV 등의 대형 임베디드 시스템에서 대용량의 멀티미디어 처리를 기존의 파일시스템으로 처리하기에는 부적절하다. 이런 이유로 ETRI에서는 멀티미디어 지원 파일시스템을 개발해 오고 있으며, 인덱싱 저장구조 설계, 저장 장치 입출력 스케줄링, 표준 POSIX API를 따르는 시스템 콜 개발 등으로 안정화시켜 나가고 있으며, 임베디드 특성에 의한 갑작스런 전원 차단의 경우 데이터의 무결성 보장을 위해 저널링 기능의 파일시스템을 통합해 나갈 계획이다.

Ⅳ. 맺음말

이상에서 임베디드 운영체제 커널 기술 동향에 대해 간단히 알아보았다. 간단히 요약하자면, 2000년대 이전의 특정 목적을 위한 임베디드 시스템용으로 각광 받았던 VxWorks, VRTX, QNX 등의 전통 전용 RTOS가 주도하던 시장에서 최근에는 임베디드 리눅스와 같은 범용 운영체제가 임베디드 시장에서 각광 받고 있는 것이 최근 추세이다. 그 주된 이유는 하드웨어와 소프트웨어 기술의 급속한 발전에 의한 임베디드 시스템의 백색가전, 정보가전 기기, 네트워크 장비, 산업용 기기 등의 광범위한 응용 시장을 가지게 됨에 따라, 아직은 연성 실시간성을 지원하고 커널의 크기가 전통의 전용 RTOS군보다 큰 것이 사실이지만, 그에 비해 다양한 기능을 갖추고 금전적, 시간적 개발 비용을 낮추고 각종 장점을 가지고 있기 때문이다.

임베디드 운영체제 기술은 향후 IT 산업의 기반이 될 유무선 통신망과 post-PC를 이용하여 유비쿼터스 환경을 이루게 할 핵심적 소프트웨어 기술로써 그 파급효과가 매우 큰 기반 기술이다. 그러나, 아쉽게도 여타의 기반 기술이 그렇듯 임베디드 운영체제의 국내 기술 수준도 국외의 기술에 의존도가 높은 것이 사실이다. 경성 실시간성이 요구되는 mission-critical한 분야에서는 기존 전통 RTOS들에 의존하고, 새로이 각광 받고 있는 유비쿼터스 컴퓨팅 환경 및 정보가전 분야에서는 국외의 기술에 아직은 미치지 못하고 있다. 그러나, 최근 불고 있는 유비쿼터스 한국이라는 말과 국내 신성장 동력산업에서 많은 연구가 이루어지고, 산업계에서도 집중 투자가 이루어지고 있는 만큼 향후 한국의 임베디드 운영체제 기술이 주도하는 임베디드 운영체제 기술 시장이 될 수 있을 것을 기대해 본다.

약어 정리

ACPI	Advanced Configuration and Power Interface
AG	Allocation Group
CELF	Consumer Electronics Linux Forum
DPM	Dynamic Power Management
DRAM	Dynamic Random Access Memory
EDF	Earliest Deadline First
FIFO	First In Firs Out
GNU	GNU's Not UNIX
JBD	Journal Block Device Layer
LILO	The Linux Loader
POSIX	Portable Operating System Interface
RTOS	Real Time Operating System
SGI	Silicon Graphics Inc.
SMP	Symmetric Multiple Processor
TRON	The Real-Time Operating System Nucleus
VOD	Video-on Demand

참 고 문 헌

[1] http://www.mvista.com
[2] http://www.tron.org
[3] Victor Yodaiken, “The RTLinux Manifesto” paper, 2000.
[4] http://www.rtlinux.com
[5] Montavista Software co., “Preemptible Linux: A Reality Check” White Paper, 2001.
[6] Clark Wiliams, “Linux Scheduler Latency,” Red Hat, Inc., Mar. 2002.
[7] 정영준, 이형석, 김홍남, “리눅스 기반 고정 스케줄링 시간을 갖는 스케줄러의 설계 및 구현,” 정보처리학회 학술논문, 2001.
[8] http://www.celinuxforum.com
[9] http://www.linuxbios.org
[10] Youjip Won and Y.S. Ryu, “Handling Sporadic Tasks in Multimedia File System,” *ACM MM Conf.*, 2000.
[11] 박진연, 송승호, 진종현, 원유집, 박승민, 김정기, “인터넷 홈서버를 위한 스트리밍 전용 파일시스템,” 방송공학회논문지, 2001. 12., pp.246-259.
[12] http://www.sgi.com/software/xfs/
[13] http://www124.ibm.com/developerworks/oss/jfs/