실시간 Linux 환경에서 상주형 게이트웨이의 설계

심 장 섭[†]·김 종 겸[††]·정 순 기[†††]

요 약

본 논문에서는 실시간 리눅스 환경에서 상주형(residential) 게이트웨이의 설계와 핵심 기능들의 구현 방법에 대한 연구를 수행하였다. 기존 실시간 운영체계의 각 기능들에 관련된 최근의 연구 결과들을 활용하여 상주형 게이트웨이의 운영체계로서 실시간 리눅스를 안정적으로 실행시킬 수 있는 장치구동기(device driver)의 개발 사례를 제시하였으며, 시스템 내에서의 성능 특성을 정성적으로 분석하여 성능을 개선시킬 수 있는 방법을 설명 하였다. 이를 통해 실시간 리눅스 환경에서 상주형 게이트웨이의 효율적인 구현 가능성을 제시하였다.

Residential Gateway Design under Realtime Linux Environment

Jang-Sup Shim[†]·Jong-Kyum Kim[††]·Soon-Key Jung[†††]

ABSTRACT

In this paper, we describe the study of residential gateway design and the implementation of its core functional features under the realtime linux environment.This paper has also suggested the developing example of device driver that can execute the realtime linux with stability based on the recent research findings of which is related to the functions of existing realtime operating system for residential gateway, and explained methods that can further improve performance by analyzing the performance characteristics of the system.And as a result, it was able to suggest the possibility of effective implementation of residential gateway under the realtime linux environment in this paper.

키워드 : 실시간 운영체계(Realtime OS), 실시간 리눅스(Realtime Linux), 상주형 게이트웨이(Residential Gateway), 접속 설정(SIP), 음성 인터넷 프로토콜(VoIP), 실시간 프로토콜(RTP), 장치 구동기(Device driver)

1. 서 론

광대역 통신망의 보급과 홈 네트워크 기술의 발달로 인해 홈 네트워크 게이트웨이 또는 상주형 게이트웨이라고 부르는 새로운 종류의 시스템 개발이 요구되고 있다.

그러나, 기존의 개발방식 대로 유닉스와 리눅스 계열의 운영 체계를 사용하면 범용적인 특성을 가지고 있으므로 특정 분야 또는 특정 기능만을 충족시키는 제품화된 성능을 제공할 수 없었으며, 시장에서 요구하는 저가의 시스템을 개발할 수 없었다[2, 7]. 또한, 시스템의 저가화 정책으로 필요 기능들의 적정 모듈화 필요성이 제기 되었는데 초기 버전으로 가전제품의 제어 기능과 인터넷을 통한 음성통화 기능이 부여된 VoIP기능을 탑재한 시스템에 대하여 모듈화 하여 구현하였다.

본 논문에서는 실시간 리눅스에 탑재 시켜 운영 할 수 있는 SIP기반 VoIP와 홈 제어용 상주형 게이트웨이의 구현 방법들에 대한 최근의 연구 결과를 중심으로 실시간 리눅스 내에서 장치 구동기의 대기 시간과 성능상의 특성을 분석하여 성능을 개선시킬 수 있는 방법을 도출 하였다. 또한 이를 기반으로 개발된 시스템을 통하여 실시간 리눅스 환경에서 구현이 가능한 상주형 게이트웨이의 소프트웨어 모델을 제안하고 샘플 소스코드와 클라이언트 화면을 제시하였다. 제2장에서는 범용 운영체계와 실시간 운영체계의 특성을 분석하였다.

제3장에서는 실시간 리눅스를 기반으로 하는 상주형 게이트웨이의 아키텍쳐를 제시하였고, 이 구조에 따라 SIP 기반의 상주형 홈 제어 어플리케이션 모델을 제안하였다. 제4장에서는 상주형 게이트웨이에 탑재된 실시간 리눅스와 응용소프트웨어의 소스코드 모듈을 제시하였고, 성능을 향상시킬수 있는 개선 사례를 제시하였으며, 제5장에서는 연구 결과와 향후 연구과제에 대하여 기술하였다.

2. 실시간 운영체계의 특성

범용 운영체계에서는 실제 프로세스가 시작되어 종료되는 시점까지의 시간을 예측한다는 것은 거의 불가능하다. 리눅스의 경우 일정한 시간 간격(interval)을 이용하여 수행

† 정 회 원 : 정보통신연구진흥원 정보화추진팀장
†† 정 회 원 : 인천기능대학 자동화시스템과 교수
††† 정 회 원 : 충북대학교 컴퓨터공학과 교수
논문접수 : 2003년 11월 5일, 심사완료 : 2004년 2월 16일

되고 있는 프로세스를 잠시 중단시킬 수 있는 메커니즘을 제공하고 있으나 시간이 종료 되었을때 즉시 중단되었던 프로세스의 재가동은 실제로 보장하지 못하며, 가상 메모리 개념을 사용하기 때문에 하나의 프로세스가 메모리에 적재되어 실행되기 전까지는 정확한 메모리의 주소를 예측할 수 없다. 이러한 현상은 윈도우 계열의 운영체계에서도 동일하게 발생한다. 또한, 실시간 개념을 이용하고 있지만 엄밀한 의미에서 하드(hard) 실시간 개념을 이용하고 있으므로 실시간 처리 프로세스에는 부적합하다. 그리고 프로세스간의 전환시 많은 오버헤드가 발생되며, 기존 유닉스 기술을 이용, 설계되었기 때문에 커널 프로세스는 비선점형 프로세스로 동작된다. 그러므로 커널 코드 내부의 위험영역(critical section)에서는 인터럽트가 불가능 하다[8].

그리고, 범용 유닉스 운영체계에서는 모든 프로세스에 동등한 컴퓨팅 시간을 할당하는 스케줄링 알고리즘을 사용하는 특성을 가지고 있기 때문에 역시 실시간 처리 프로세스에는 부적합 하다.

범용 유닉스나 리눅스와는 달리 실시간 리눅스에서는 정확한 타이밍과 시간예측이 가능하다. 실시간 리눅스는 범용 리눅스 내에 작고 간단한 또 하나의 실시간 운영체계를 가지고 있다. 범용 리눅스는 실시간 태스크가 수행되고 있지 않을 경우에만 자신의 태스크를 수행할 수 있으며, 실시간 태스크가 CPU 자원을 요구하게 되면 수행되던 범용 리눅스의 태스크는 실시간 태스크에 의해 CPU 자원을 선점 당해 중단 된다.

실시간 리눅스가 동작하면 범용 리눅스 시스템은 실시간 커널에 의해 제어된다. 실시간 커널은 기본적으로 비선점형이며, 범용 리눅스 커널 보다 작고 빠르다. 실시간 커널에서의 프로세스 스케줄링 시간은 최대 20usec를 넘지 않는다. 따라서 리눅스를 기반으로 하는 실시간 운영 체계의 가장 큰 장점은 개발 비용이 적게 든다는 점이다. 현재 많은 실시간 운영체계(RTOS : Real-time Operating System)가 공존하고 있지만 실제로 시스템에 탑재할 수 있는 RTOS는 많은 로열티를 요구하고 있다. 단순히 연구 및 개발 목적으로 RTOS를 이용할 경우는 실시간 리눅스가 가장 적합한 OS로 분석 되고 있다[6].

본 논문에서 제안하는 상주형 게이트웨이 모델은 기본적으로 다양한 하드웨어 장치들을 제어하거나 장치의 기능을 이용 하는데 목적을 두고 있기 때문에 무엇보다도 짧은 문맥 전환 시간(context switching time)이 필요하였다. 그리고 어느 특정 장치에 의해 CPU가 선점되거나 예상치 못한 에러 발생 등의 문제점을 최소화 하기 위해서 실시간 리눅스 커널을 이용하는 것을 전제로 하였다[9].

특히 홈 제어를 목적으로 하는 상주형 게이트웨이의 경우 가정내의 다양한 장치들과 연결되기 때문에 해당 장치의 제어, 메시지 송수신 등과 같은 기능을 수행하기 위해서는 무엇보다도 응답시간이 짧아야 한다. 따라서 각 이벤트에 대한 응답 시간을 최소화 시키기 위해서는 상주형 게이트웨이의 내부 태스크는 실시간으로 작동되어야 한다.

또한 멀티미디어 데이터와 같은 서비스들은 실시간 컴퓨팅을 필요로 하며 특히 IEEE1394 네트워크의 관리시 IEEE1394버스의 리셋(reset)과 네트워크 에러에 대한 리포팅(reporting)도 실시간으로 처리되어야 한다. 구현된 실시간 리눅스 커널은 이들 기능의 처리를 가능케 한다.

본 논문에서 제안하는 상주형 게이트웨이는 기 연구에서 사용된 범용OS의 제한을 극복하기 위하여 실시간 리눅스 운영체계를 기반으로 하였으며 시스템의 개발시 고려된 주요특성들은 다음과 같다.

2.1 멀티 태스킹

윈도우 CE의 경우 단지 8개의 우선순위 레벨을 제공하고 있다. 따라서 실시간 응용 태스크를 구현하는데 많은 제약이 따른다. 대부분의 경우 실시간 기능의 구현에 적합한 우선순위 레벨은 32개로 알려져 있다[5].

그러나 VxWorks는 256개의 우선순위 레벨을 가지고 있으며, 리눅스는 256개의 우선순위 레벨과 RR(Round Robin), Quantum, FCFS(First Come First Service) 스케줄러 등을 위한 우선순위를 가지고 있다[8]. 리눅스는 이처럼 다양한 레벨의 우선 순위를 제공하고 있으므로 프로세스에대한 자원 할당이 효율적으로 이루어 질 수 있으며, 이같이 멀티 태스킹이 가능한 특성을 구현에 활용하였다.

2.2 동기화

동기화는 실시간 태스크를 실행할 때 상호 독립적인 자원(디바이스, 기억장치, 버퍼 등)들을 공유할 수 있게 하며, 자원들간에 충돌 발생을 방지하는 특성을 가지고 있다. 이러한 특성은 태스크 종속성(A 태스크에서 명령어 y를 실행한 다음 B 태스크에서 명령어 x를 실행)을 구현하는 데 필수적인 조건이 된다.

사용자는 대기기능, 객체 동기화, 연동기능 및 메시지를 사용하여 스레드(thread)의 동작을 제어 할 수 있다. 세마포어(semaphore)를 이용하는 기존 의 해결 방식은 무제한 우선순위 변환(unbounded priority inversion)으로 간주할 수 있다. 우선순위 변환은 우선순위가 높은 태스크와 우선순위가 낮은 태스크간에 실행 순서가 바뀔 때 발생하는 현상이다. 이러한 우선순위 변환은 중간 정도의 우선순위를 갖는 태스크가 하위 우선순위를 갖는 태스크를 대신할 때도 발생될 수 있다. 이러한 문제에 대한 기존의 해결 방식으로는 SPIP(Simple Priority Inheritance Protocol)와 CPCP(Complex Priority Ceiling Protocol)를 들 수 있다. 이들은 일반적으로 HLP(Highest Locker Protocol)기법으로 구현되고 있다. CPCP의 구현에는 많은 비용이 소요되며, 또한 제한

적인 기능만을 제공한다. 따라서 대부분의 RTOS에서는 SPIP 방식을 채택하고 있으며, 본 논문에서도 시스템의 구현에 SPIP 방식을 사용하였다.

2.3 인터럽트와 이벤트 핸들링

실시간 어플리케이션은 인터럽트를 사용하여 적절히 외부 이벤트에 응답한다. 리눅스, QNX 및 VxWorks등은 선점 인터럽트 서비스 루틴(preemptible interrupt service routine)을 이용하여 인터럽트와 각종 이벤트를 처리한다. 윈도우 CE에서는 ISR(Interrupt Service Routine)을 이용하여 우선순위 0단계(최상위 우선순위)에서도 작동하는 IST(Interrupt Service Thread) 방식을 사용한다. 따라서 윈도우 CE에서는 인터럽트를 ISR과 IST의 두 단계로 분리시켜 적절히 처리한다.

본 논문에서 적용한 실시간 어플리케이션 인터페이스(RTAI : Real Time Application Interface)는 실시간 하드웨어 추상계층 (abstraction layer)을 통하여 8259 인터럽트 제어기를 처리하는 방식으로서 이벤트 처리에 대한 인터럽트를 매우 근본적인 방식으로 구현 처리하였다.

2.4 프로세스간의 통신

QNX는 프로세스간의 통신(IPC : Inter-Process Communication)에 기존 소켓(socket) 보다 작은 메모리 공간을 사용하는 강화된 소켓을 사용한다.

리눅스는 프로세스간의 통신을 위해 자체에서 풍부한 IPC 집합을 제공하고 있지만, VxWorks는 분산된 시스템 처리를 위해 원격 프로시저 호출(RPC : Remote Procedure Call)을 지원한다.

본 논문에서는 RTAI를 사용하여 실시간 태스크 간에 또는 실시간 태스크와 리눅스 태스크간의 통신에 FIFO(First In First Out) 버퍼만을 제공하는 방식을 사용하였다.

2.5 타이머와 클럭

본 논문에서 제안된 RTOS는 타이머, 시간 트리거 태스크(time triggered task) 및 클럭 기능의 구현이 가능하다. RTOS를 탑재한 시스템에서 본 소프트웨어에 응답하도록 하드웨어를 설계하여 nano-second 단위의 클럭을 발생시켜 사용하였다.

2.6 메모리 관리

과거 대부분의 RTOS는 프로세서내 메모리 관리 장치(MMU : Memory Management Unit)의 부족으로 인한 비결정성 특성 때문에 가상 메모리 지원에 대한 필요성을 느끼지 못하고 있었다. 또한 동적 메모리 할당(DMA : Dynamic Memory Allocation) 방식은 프로그래밍상의 유연성은 제공하지만 불필요한 데이터 수집에 많은 오버헤드를 초래 하였다. 본 논문에서는 DMA 방식을 제한적으로 사용하였다.

3. 상주형 게이트웨이의 모델

3.1 시스템 아키텍쳐

댁내 홈 제어 및 SIP기반 VoIP용 게이트웨이 시스템 용으로 본 논문에서 제시하는 상주형 게이트웨이에서 태스크의 실시간 처리를 위해 고려한 사항은 다음과 같다.

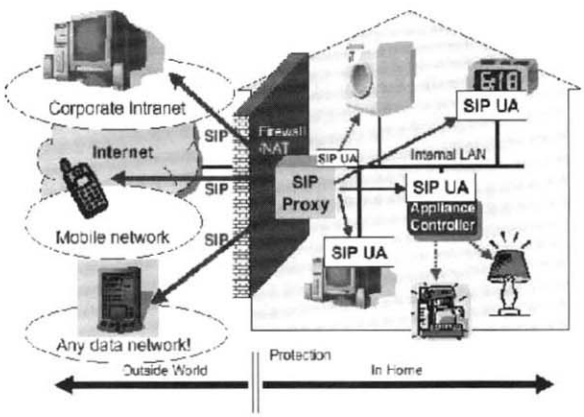
첫째, 이벤트의 처리는 기본적으로 소프트웨어가 담당한다. 이는 실시간 태스크의 기본적인 스케줄링이 이벤트에 기반 한다는 것을 의미하며, 발생된 이벤트에 의해서 해당 태스크가 처리되어야 한다는 것을 의미한다.

둘째, 실시간 컴퓨팅을 제공하기 위해서는 처리 시간에 기반한 각 태스크들의 조정에 우선순위 중심의 스케줄링이 필요하며, 우선순위가 낮은 태스크에 할당된 CPU 자원의 낭비를 방지하기 위해서 실시간 커널에 의한 CPU 선점이 필요하였다. 이를 위해서 우선순위를 조정할 수 있는 특정한 알고리즘을 적용 하였다.

마지막으로, 상주형 게이트웨이를 전체 시스템의 관점에서 보면, 실제로 실시간 처리가 중요한 부분과 그렇지 않은 부분이 혼재하기 때문에 두 가지 종류의 태스크를 모두 지원할 수 있어야 한다.

그러므로 본 논문에서는 상위 소프트웨어를 실시간 태스크와 비실시간 태스크가 공존하는 하나의 통합된 프레임워크로 구성하여 원하는 시간내에 중요한 기능들이 빠르면서도 제외됨이 없이 수행 되도록 구현한 것이 기존 시스템 대비 진보된 특징이다.

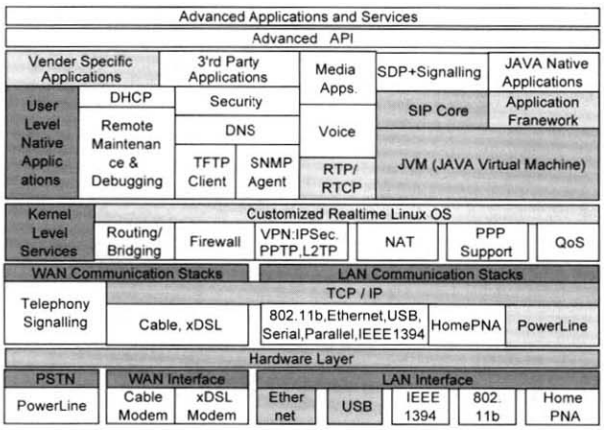
개발환경으로는, 리눅스커널(2.4.18 버전)과 JAVA runtime 환경(1.3.1버전), OSGI framework, SIP Proxy, SIP-X10 bundle module 등 소프트웨어 환경과 RedHat Linux(7.2 버전)를 탑재한 셋탑박스, X-10 스위치모듈, 메신져서버 등 제어용 하드웨어를 구축하였으며, 그 상위에 본 연구를 통하여 구현된 응용 소프트웨어를 탑재하여 기능을 구현하였다.



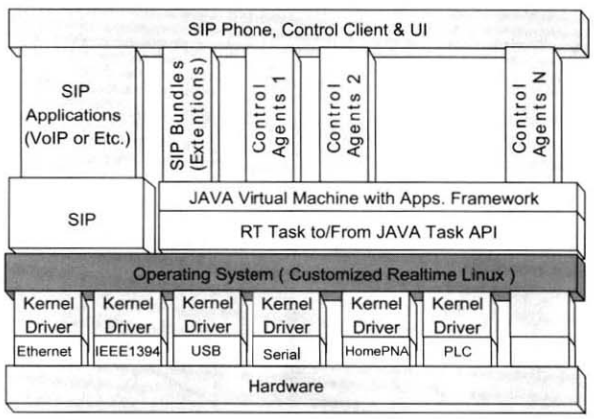
(그림 1) SIP를 포함하는 상주형 게이트웨이

(그림 1)에서 실제 네트워크 상에서 구현되고 시험된 상주형 게이트웨이의 다양한 응용과 네트워크 내에서의 물리적 위치를 표현 하였으며, (그림 2)는 사용자 규격에 의해 결정되는 기능들간에 상호작용 하는 요소들의 관계를 계층적으로 표현하였다.

IP 네트워크와의 접속을 위해 게이트웨이는 운영 체계의 커널에서 제공하는 TCP/IP상에서 구현된 기능을 이용하며, PSTN과의 접속은 게이트웨이에 내장된 전화 신호(telephony signaling)를 처리하는 하드웨어와 해당 장치 구동기를 통하여 이루어진다[7]. 또한 PSTN망과의 직접적인 음성신호의 연결 뿐만 아니라 SIP, RTP/RTCP 프로토콜을 이용하는 VoIP 계층을 통하여 PC-to-PC 혹은 PC-to-Phone으로의 연결도 가능토록 구현하였다.



(그림 2) 상주형 게이트웨이의 기본구조

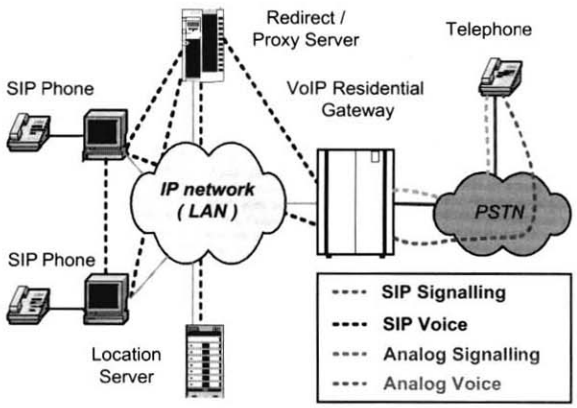


(그림 3) 상주형 게이트웨이의 계층구조

VoIP 기능을 수행하는 부분으로는, TCP/IP 계층을 경유하여 음성 송수신 세션을 관장하는 SIP 및 SDP 등이 있고, 실제로 음성 매체의 전달과 처리를 담당하는 RTP/RTCP로 구성된 사용자 영역의 기능들이 있으며, 제어 부분으로는 크게 LAN 인터페이스에 해당되는 각종 인터페이스들과, 외부 네트워크 과의 연결에 사용되는 WAN 인터페이스로 구성되는 하드웨어 계층, 실시간 커널이 제공하는 각각의 장치구동기 및 어플리케이션 프레임워크와의 상호 작용을 통하여 사용자에게 제어 UI(User Inter-face)를 제공하는 어플리케이션과 서비스가 내장 되어 있다. 계층적으로 구현된 소프트웨어는 (그림 3)과 같은 구조로서 사용자영역(어플리케이션 영역)과 운영체계의 커널영역 및 하드웨어영역으로 모듈화 되어 있다.

3.2 SIP-VoIP 어플리케이션 모델

인터넷텔레포니(telephony)는 VoIP 기술을 이용하여 인터넷상에서 전화서비스 및 전화 응용서비스의 구현을 목표로 하였다.



(그림 4) VoIP 어플리케이션 기능 모델

VoIP용 상주형 게이트웨이를 통해 인터넷망에 접속된 VoIP폰과 PSTN폰 상호간에 음성통화가 가능해야 한다. 이러한 기능을 위해서 네트워크상에 SIP폰, 프락시서버/방향전환 서버 및 위치서버등과 같은 요소 시스템들이 배치되어야 한다[7].

본 논문에서 구현된 상주형 게이트웨이는 SIP폰(터미널) 기능을 위해 VoIP 모듈이 탑재 되었으며, PSTN 상호간에 음성호를 송수신하는 역할을 담당한다. VoIP용 상주형 게이트웨이와 다른 시스템 요소들은 테스트베드 차원으로 (그림 4)와 같이 구축 되었으며 이를 통해 기능과 성능을 확인하였다

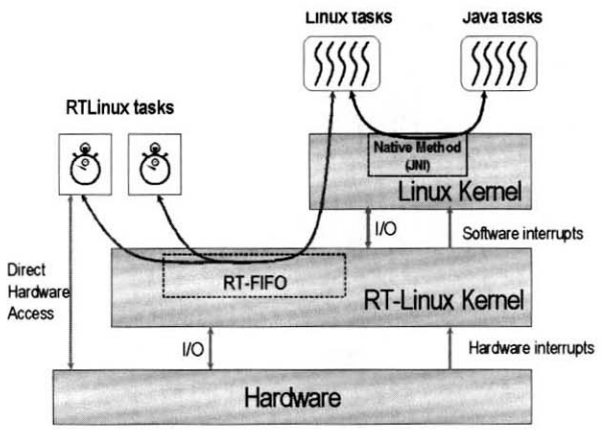
3.3 홈 제어 어플리케이션 모델

제시한 상주형 게이트웨이는 홈 제어에 필요한 여러 종류의 인터페이스 장치를 관리하며, 사용자가 장치를 직접 핸드링하는데 필요한 UI와 응용 프로그램을 관리하는 모듈로 구성되었다.

3.3.1 어플리케이션 프레임워크

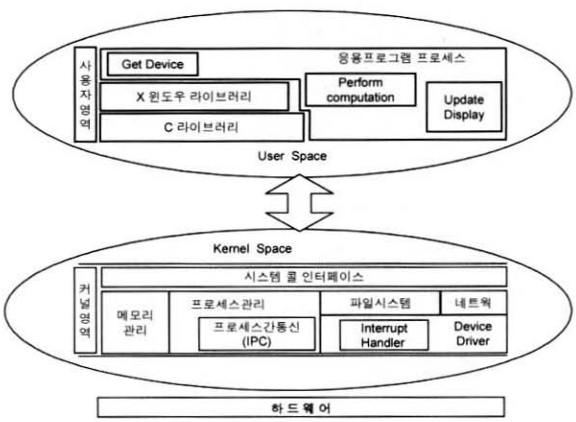
어플리케이션 프레임워크의 기본적인 동작환경은 Java VM이며, 각 제어 중계자(Agent)들은 리눅스 커널의 실시간 태

스크에 이용하는 RT-FIFO를 사용한다. (그림 5)에서 실시간 커널, Java 태스크 및 리눅스 태스크 상호간의 관계성을 제시하였다. 제어 중계자는 Java 태스크들로 구성되지만 실시간으로 동작되지는 않는다. 다만 실시간 커널에서 수행되는 RT-태스크와의 연동을 위해서 Java 태스크와 RT-태스크간의 동기화가 필요하다. 따라서 RT-태스크와 Java 태스크 상호간의 동기화 과정을 실시간 커널에서 수행 하였다.



(그림 5) RT 태스크와 Java태스크간의 상호관계

시스템의 구조적 구성으로는 (그림 6)과 같이 구현하여 실시간 어플리케이션이 동작하였다.



(그림 6) 사용자영역과 커널 영역간의 상호관계

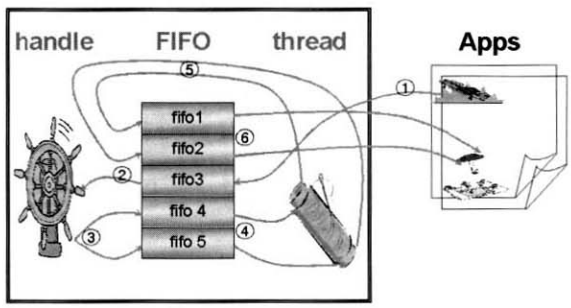
실시간 어플리케이션 기능은 주로 사용자 영역에서 실행된다. Daemon을 포함한 모든 사용자 영역의 프로그램은 실시간 성능을 평가하는데 중요한 역할을 한다. 특정한 사용자 태스크를 수행하는 도중에 우선순위가 높은 실시간 태스크가 시스템을 호출하게 되면 수행 중인 사용자 태스크는 지연(suspend)될 수 있다. 사용자영역내에 포함된 Get Device Data, Perform Computation 및 Update Display와 같은 명령문은 POSIX 스레드나 리눅스 프로세스로 구현하였으며, 이들은 최적화 모델이 아닌 시제품(prototype) 차원에서 구현하였다.

3.3.2 RT와 Java 태스크간의 동기화

RT태스크에서 Java 태스크로의 동기화 과정에는 다음과 같은 요소들이 이용되었다.

- RT FIFO
- 공유 메모리(shared memory)
- 상호 배제(mutual exclusion) 기능
- 실시간 리눅스 스레드의 기동과 일시 중단 기능들이 있다.

(그림 7)에서와 같이 RT 태스크에서 Java 태스크 로의 동기화 과정에는 실시간 FIFO큐를 이용하였다. 큐는 기본적으로 한쪽 방향으로만 접근이 가능하지만 두개의 FIFO 큐를 사용하여 양방향에서의 큐 접근도 가능하도록 하였다.



(그림 7) RT와 Java 태스크간의 동기화

즉, 모든 메시지와 이벤트는 실시간 FIFO를 경유하여 커널로 전달되기 때문에 커널 모듈 내부의 핸들러와 스레드 가이러한 형태의 태스크를 제어하게 된다. Java 태스크에서 RT 태스크로의 동기화 과정도 위에서 설명한 내용과 유사하게 진행된다. 즉, 이러한 형태의 동기화는 모듈에 의해 생성, 관리되는 실시간 FIFO에 기반하는 메시지 큐를 실시간 커널이 적절하게 이용 함으로서 두 태스크간의 동기화를 가능하게 하였다.

4. 상주형 게이트웨이의 소스코드 모듈

본 논문에서 제안하는 상주형 게이트웨이의 응용 소프트웨어 소스코드 모듈과 알고리즘의 주요 부분은 다음과 같으며, 개발되어 구현이 검증된 Web-base 클라이언트 제어 화면을 (그림 9)에 제시하였다.

4.1 모듈의 적재와 하적

Init init_module() ;

위 함수는 모듈을 커널에 적재(load)할 경우 호출된다. 적재에 실패(Failure)할 경우에는 '0'값을 리턴한다.

Void cleanup_module( ) ;

위 함수는 모듈을 커널로부터 하적(unload)할 경우에 호출된다.

4.2 스레드 스케쥴링

스케쥴링을 위해 실시간 리눅스는 특정한 시점에만 구동되는 스레드 코드를 요구하는 방법을 사용하였다. 또한 실시간 리눅스는 스레드 스케쥴링을 위해서 몇가지 종류의 클럭을 제공하며 본 설계에서는 현재 시간을 판독, 이용하기 위해서 함수 gettime( )을 사용하였다.

int clock_gettime(clockid_t, clock_id,
*struct timespec *ts) ;*

clock_id는 판독후에 사용되는 클럭을 의미한다. ts는 판독하여 사용할 클럭을 저장하는 장소를 의미한다.

Struct
*timespec time_t, tv_sec ; /*seconds*/*
long tv_nsec / nanoseconds */ ;*

사용할 클럭의 종류는 다음과 같이 구분할 수 있다.

CLOCK_UST :

UST는 고정시스템 시간(Unadjusted System Time)을 의미하며, 이 클럭은 재조정과 초기화가 절대로 불가능하다.

CLOCK_REALTIME :

REALTIME은 표준 POSIX 실시간 클럭을 의미한다.

CLOCK_8254 :

비다중프로세서(non-multiprocessor)에서 스케쥴링 처리에 사용된다.

CLOCK_APIC :

CLOCK_APIC는 대칭형 다중 프로세서(Symmetricmulti-processors) 시스템에서 사용되며, clock_ gettime()을 실행하는 프로세서의 로컬 APIC(Local Advanced Programmable Interrupt Controller) 클럭에 해당된다. 외부의 I/O APIC 제어기와 함께 프로세서간의 통신 등에도 사용된다.

스레드 우선순위(thread priority)는

pthread_attr_setschedparam(3) 또는
pthread_setschedparam(3)
int pthread_setschedparam
(pthread_t thread, int policy, const struct
sched_param param) ;

를 사용하여 스레드의 생성시간을 변경 할 수 있다.

4.3 실시간 FIFO 큐의 사용

실시간 FIFO는 리눅스 프로세스와 실시간 스레드에 의해 읽기와 쓰기를 수행하는 단 방향의 선입선출 큐이다. 실시간 FIFO의 초기화에는 다음과 같은 코드를 사용하였다.

#include<rtl_fifo.h>
int rtf_create (unsigned int fifo, int size) ;
int rtf_destroy (unsigned int fifo) ;

rtf_create( )는 FIFO를 위해 특정 크기의 메모리 버퍼를 할당한다. *"fifo"*파라메터는 시스템의 장치 파일의 Minor 번호를 의미하며

rtf_destroy( )는 버퍼의 제거에 이용되었다.

4.4 공유 메모리의 사용

물리적인 페이지 블록에 대한 이미지를 적재할 때 운영체계는 페이지를 할당해야 하며, 이미지의 실행이 끝나고 하적 될 때 페이지 블록을 해제해야 한다. 따라서 공유 메모리의 사용시 다음과 같은 블록 할당과 해제 함수를 사용하였다.

#include
*void * mbuff_alloc (const char *name, int size) ;*
*void mbuff_free (const char *name, void * mbuf) ;*

함수 mbuff_alloc( )이 호출되면 공유 메모리에 특정 크기의 블록이 할당되며, 블록의 계수기는 1로 세팅된다. 블록 할당에 성공하면 새로 할당된 블록의 포인터 값이 전달(return)되며, 실패했을 경우 널(Null) 값이 전달된다.

함수 mbuff_free( )는 특정 버퍼로부터 mbuf를 해제하며, 기준 계수기의 값을 1씩 감소 시킨다. 따라서 계수기의 값이 0 이 되면 버퍼는 완전히 해제된다.

4.5 타이머 사용

외부 장치로부터의 인터럽트를 사용하는 대신에 내부 시스템의 타이머를 사용하여 특정한 연산 (operation)을 효율적으로 스케줄링할 수 있다[4]. 리눅스에서 사용하는 타이머는 일반적으로 10msec의 지연시간을 가지기 때문에 특정한 실시간 연산을 수행 하기에는 너무 느리다. 따라서 리눅스 커널 내부의 타이머를 실시간 응용에 적합하도록 약 1 usec 단위로 동작되도록 조정하여 사용하였다.

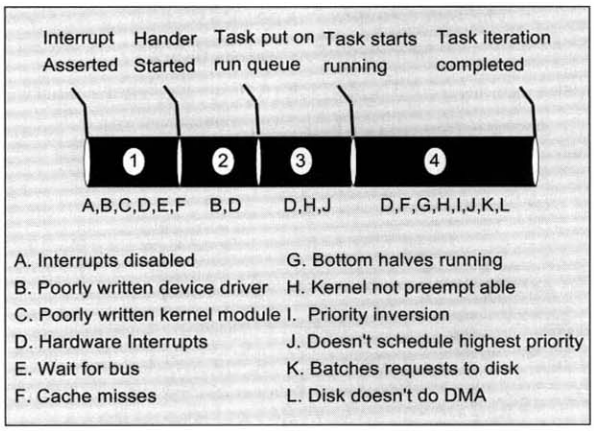
4.6 태스크 수행과정의 개선

(그림 8)은 기존의 실시간 태스크 수행과정에서 시간축(time line)을 나타내며, 인터럽트의 발생, 핸들러의 구동, 큐

에 태스크 적재, 태스크 시작, 태스크의 순차적인 완료 순서로 진행된다[4].

CPU를 통해 시스템의 태스크가 처리되며, 이러한 과정 이전의 태스크는 장치호출 명령에 의해 커널에서 대기(blocked)하며, 새로운 데이터의 도착을 알리는 메시지를 기다린다.

본 논문에서는 위와 같은 실시간 태스크의 수행과정 중에 개선시킬 수 있는 핸들러 구동, 태스크의 실행과 태스크 실행의 유지 과정부분을 다음과 같이 개선하였다



(그림 8) 실시간 이벤트의 시간 축

4.6.1 핸들러 구동

x86 구조에서 인터럽트 핸들러는 일반적으로 약 15 usec 내에 구동 된다.

핸들러가 다른 장치와 동시에 동작하지 못하는 것으로 판단되거나, 공유 데이터의 특정 항목(item)에 배타적인 접근을 요구하는 커널의 특정 부분을 만나게 되면 인터럽트는 불능(disable) 상태가 된다. 또한 리눅스 커널이 해당 핸들러가 적절한 시점이 될 때까지 재진입할 수 없다는 것을 확인할 경우에도 인터럽트는 불능 상태가 된다. 리눅스는 시험을 통해 100msec 이상 불능 상태에 있는 인터럽트를 확인할 수 있는데, 이는 실시간 장치의 핸들러가 그 시간만큼 지연되었다는 것을 의미한다. 또한 실시간 작업 중에 발생하는 인터럽트를 처리할 수 없다고 가정하면, 시스템의 구조 자체에서 인터럽트 핸들러의 실행을 지연 시키는 원인을 제공한다고 간주할 수 있다. 이러한 두 종류의 시간 지연요소는 인터럽트에 대한 장치 간의 정보교환에 소요되는 시간과 인터럽트 핸들러를 위한 코드와 데이터를 캐쉬 메모리로 적재 시키는데 소요되는 시간을 의미한다. 인터럽트를 불능시킬 수 없거나 시스템이 다수의 인터럽트를 처리할 수 없는 경우는 인터럽트 처리기는 수십 usec 내에 재실행될 수 있다. 본 시스템에서는 처리가 불가능한 인터럽트 문제를 리눅스 코드로 재코딩하여, 코드 내에서 허용할 수 있는 인터럽트의 불능상태 지원 및 인터럽트 발생시 커널 동작의 금지를 통해 해결토록 하였다.

4.6.2 태스크 실행

커널의 선점 문제는 특히 시간 축의 진행을 위해 매우 중요하다. (그림 8)에서와 같이 특정 태스크가 호출되고, 실행준비가 완료된 상태에서 다른 인터럽트가 발생했을 때 커널은 시스템의 호출 절차를 진행중에 있을 가능성이 있다. 따라서 커널은 두 작업간의 스위칭 처리를 지연시킬 수 밖에 없었다. 표준 리눅스 커널은 위와 같은 시간지연 상황을 개선하기 위해서 패치(patch) 없이 약 100msec 단위로 대기시간을 제공하고 있으며 작업 스케쥴링 문제는 이러한 측면에서 매우 중요 하다. 또한 리눅스는 평균 시스템 처리량(average system throughput)에 알맞게 최적화 되었기 때문에 우선순위가 높은 작업이라도 지연될 수 있었다. 본 시스템에서는 리눅스 코드의 변경을 통하여 지연을 최소화 시켰다.

4.6.3 태스크 실행의 유지

작업이 일단 시작되면 태스크의 실행은 계속 진행되어야 한다. 그러나 시스템 호출로 작업이 지연될 수 있으며, 또한 하드웨어에서 인터럽트가 발생하면 태스크의 실행속도가 늦어질 수 있다. 캐시 메모리의 누락(fault)이나 페이지 장애와 같은 심각한 장애를 방지하기 위해서는 추가의 패치 프로그래밍이 필요하였다. 커널 역시 태스크를 수행하는 대신에 Bottom Half를 포함한 인터럽트 핸들러를 가동할 수 있도록 조치 하였다.

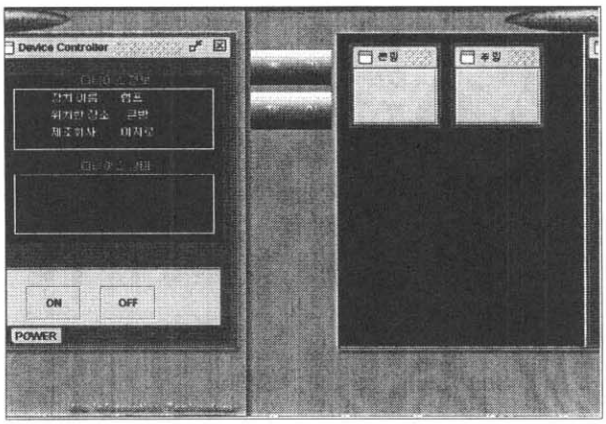
자원(resource)상에 세마포어나 Mutex와 같은 태스크 블록이 존재할 경우 우선 순위 변환방법을 채택하여 작업을 잠시 동안 차단시킬 수 있었다.

또한, 작업 지연 문제는 패치를 통한 코드의 일부수정을 통하여 해결하였으며, 우선순위 변환이 발생되지 않는 동기화 방식을 채택하였고 실시간 지원이 가능한 장치 구동기의 사용을 통하여 기능을 개선하였다.

4.7 클라이언트 제어 기능 화면

이상과 같이 구현된 클라이언트 제어기능은, 상주형 게이트웨이와 원격으로 연결되어 댁내에 설치되어 있는 장치들을 사용자가 제어할 수 있도록 환경을 만들어주는 역할을 한다. 예를 들면, 웹을 이용하는 형태의 장치접근 기능으로서 클라이언트 PC에서 직접 실행되는 독립적인 프로그램이다. 이를 통하여 사용자는 상주형 게이트웨이의 제어 중계자를 접근하도록 구현하였다.

이번의 결과로서 (그림 9)와 같이 상주형 게이트웨이에서 직접 동작하고 원격지 PC에서 실행되는 독립적인 시스템을 구현하여 그 기능과 성능을 확인하였다.



(그림 9) 상주형 게이트웨이 상의 Client 화면

5. 결 론

본 논문에서는 실시간 리눅스의 응용 모델을 제시하기 위하여 실시간 리눅스와 범용 실시간 운영체계의 주요 특성을 요소별로 비교하였으며 구현된 시스템의 요소 프로그램과 클라이언트 화면을 제시하였다. 이를 통하여 실시간 리눅스의 응용 모델로서 SIP 기반의 VoIP용 상주형 게이트웨이 시스템을 제시하였고, 구현된 기능은 주로 실시간 리눅스 커널상에서 실행되는 시간대기 기능과 장치 구동기로 구성되었다. 작업 처리에 소요되는 대기시간을 분석하여 범용의 실시간 장치 구동기의 성능과 비교하여 기능을 제시하였다. 이를 통하여 실시간 리눅스 상에서 작업 처리에 사용할 수 있는 실용적인 모듈을 도출하였다.

본 연구와 관련되어 향후 과제로는 차세대 통신 프로토콜과 같은 응용 소프트웨어와의 연동기능, 대기시간의 단축 방법 및 효율적인 실시간 처리 프로그래밍 기법의 개선 방안들이 연구될 수 있다.

참 고 문 헌

[1] 손경호, 송재원, "홈네트워크 관리를 위한 SIP 및 OSGI 연동 시스템", 한국정보과학회 학술논문집. Vol.29, No.2, Oct., 2002.

[2] B. Weinberg, Technologist, "Building Internet Appliances with Linux," Apr., 2000, http://www.fsmlabs.com.

[3] Jong-Koo Lim, "Synchronization between RT-tasks and Java-tasksin Embedded Real-time Systems," RT systems Lab., KNUT, Mar., 2002.

[4] K. Dankwardt, Founder and President of K Computing, "A Simple model of Realtime Applications," Dec., 2000, http://www.linuxdevices.com/articls/AT5709748392.html.

[5] M. Bar, "Linux Internals", Osborne Publ. pp.21-77, 161-210, 219-232, 2000.

[6] M. Barabanov, "Getting started with RealTime Linux," 1999(baraban@fsmlabs.com), http://alpha400.ee.unsw.edu.au/elec2042/linux/GettingStarted.txt.

[7] M. Lad, M. Jalan, D. Patil, S. Sule, "SIP based Single-line VOIP Residential Gateway," 2002, http://www.ee.iitb.ac.in/uma/~ncc2002/proc/NCC-2002/pdf/n135.pdf.

[8] R. Yerraballi, Dept of Computer Science and Engineering, Uni. Of Texas at Arlington, "Real-time OS : An Ongoing Review," Apr., 2000, http : //www.cse.ucse.edu/~sbrandt/tss2000/proceedings/16.pdf.

[9] Yoshiaki Takabatake, Ichiro Tomoda "Home Gateway Architecture and its Implementation," IEEE Transactions on Consumer Electronics, pp.1161-1166, ISSN 0098-3063.

심 장 섭

e-mail : hl1iit@chol.com

1986년 한양대학교 전자공학과(학사)

1992년 한양대학교 전자공학과(석사)

2003년 충북대학교 컴퓨터공학과 (박사과정 수료)

2001년 데이콤 종합연구소 지능망개발팀장

2003년 ㈜위즈정보기술 연구소장/이사

2003~현재 정보통신연구진흥원 정보화추진팀장

관심분야 : DBMS, 임베디드 SW, 통신프로토콜

김 종 겸

e-mail : jkkim@kopo.or.kr

1992년 청주대학교전자공학과(석사)

2002년 충북대학교 컴퓨터공학과 (박사과정 수료)

1993~현재 인천기능대학 자동화시스템과 부교수

관심분야 : DBMS, 실시간시스템

정 순 기

e-mail : soonkey@chungbuk.ac.kr

1982년 Dortmund대학 전산학과, Dipl.Inf.

1994년 Groningen대학 전산학과, Dr.

1994년 충북대학교 전자계산소장

1998년 한국과학재단 한독기초과학 정보분과위원장

2000년 충북대학교 도서관장

1985~현재 충북대학교 컴퓨터공학과 교수

관심분야 : DBMS, 실시간 시스템, S/W 공학