리눅스 기반 실시간 처리 VoIP 단말기 시스템의 설계 및 구현

이 명 근[†] · 이 상 정[††] · 서 정 민[†††] · 임 재 용[†††]

요 약

본 논문에서는 리눅스를 기반으로 실시간 음성 처리 VoIP 단말기를 설계 구현한다. 설계 구현하는 하드웨어 시스템은 i486 프로세서를 기반으로 설계되며, 음성 코덱칩을 사용하여 실시간으로 음성 데이터를 처리한다. 또한 실시간 음성 데이터를 관리하고 처리하기 위해 리눅스 기반 실시간 처리 운영 체제인 RTLinux를 포팅하여 실시간 음성처리 모듈을 구현한다. 음성처리에 사용한 음성처리 모듈은 ITU-T 표준 음성 코덱인 G.723.1 사용하여 30ms 내에 24바이트로 인코딩/디코딩된 음성 데이터를 전송하도록 하고, 음성 전달의 QoS를 보장해 주기 위해서 리눅스에 실시간 음성처리 디바이스 드라이버를 설계 구현한다. 설계 구현하는 시스템의 테스트 및 타당성 검증을 위해 음성채팅 응용 프로그램을 단말기에 구현하여 통화품질을 시험한다.

A Design and Implementation of the Real-Time VoIP Terminal System Based on Linux

Myoung-Kun Lee[†] · Sang-Jeong Lee[††] · Jung-Min Seo[†††] · Jae-Yong Lim[†††]

ABSTRACT

In this paper, a VoIP (Voice on Internet Protocol) terminal system, which can process voice in real time based on Linux, is designed and implemented. The hardware of it is designed using a i486 processor and a DSP codec chip which encodes and decodes voice data in real time. As an operating system, RTLinux, which is a real-time operating system based on Linux, is ported to manage real-time voice processing. The voice processing module of the system uses G.723.1 voice codec of ITU-T standard. It transfers voice data within 30ms to assure good voice quality. In order to satisfy the real time requirements and QoS (Quality-of-Service) for the voice data, the real-time voice processing device driver is designed and implemented. To verify the system, the chatting application program is developed and tested for QoS of the system.

키워드 : 리눅스(Linux), 실시간 처리(Real Time), VoIP

1. 서 론

Real-Time OS의 종류는 그 종류만 해도 그 수를 헤아리기 힘들다. 일반적으로 많이 알려진 Real-Time OS들을 열거하면 지금은 WindRiver와 통합된 ISI의 pSOSystem, WindRiver의 VxWorks, 마이크로텍의 VRTX, 마이크로웨어의 OS-9 등의 상용 Real-Time OS와 ECOS, RTLinux 등의 비상용 Real-Time OS, 교육용으로 나온 uCOS Real-Time Kernel 등이 있다. 본 논문에서는 오픈 소스(버그에 대해 빠른 수정가능), 가격 경쟁력(No royalty), 품질 경쟁력(다양한 서비스 제공), 확장성 및 이식성(다양한 프로세서 지원, 다른 기종간의 이식 수월), 안정성 및 신뢰성(다수의 사용자가 이용하는 안정되고 검증된 운영체제) 등의 장점을 갖는 일반 리눅스 커널을 기반으로 Real-Time 기능을 커널에 추가하는 RTLinux를 사용하여 VoIP 시스템에서의 Real-Time OS로서 리눅스 사용 가능성과 성능 등을 검토한다.

유닉스 계열 운영체제는 시분할 스케줄링 방식을 사용하도록 고안되었으며 리눅스 스케줄러는 프로세스들의 우선순위를 주기적으로 재계산하여 각 프로세스들이 프로세서를 공평하게 공유할 수 있도록 만들어졌다. 그러나 VoIP 시스템 등과 같이 음성처리 목적으로 고안된 임베디드 시스템의 경우 이러한 비실시간 스케줄링 방법으로는 임베디드 시스템의 특수 목적을 만족하기 어려운 실정이다[5, 13]. 이러한 실시간 시스템 운영체제를 직접 구현하는 것보다는 기존의 유닉스 계열의 운영체제를 실시간 시스템에 사용할 경우 많은 장점이 존재한다. 특히 리눅스와 같이 공개된 운영체제를 사용하여 새로운 실시간 운영체제의 기반이 아닌

† 정 회 원 : 순천향대학교 대학원 전산학과
†† 정 회 원 : 순천향대학교 정보기술공학부 교수
††† 정 회 원 : 주)다이알로지코리아 연구소
논문접수 : 2001년 10월 4일, 심사완료 : 2001년 12월 22일

기존 운영체제의 확장으로 실시간 시스템을 구성한다면, 개발자들에게 익숙한 개발환경을 제공함으로써 개발 시간을 많이 단축할 수 있다[18-20].

일반적인 운영체제에서는 선점형 멀티태스킹 시스템이다. 이러한 체제에서 정확한 시간주기를 가지고 일을 해야하는 프로세서에게 커널이 정확한 타이밍을 맞춰준다는 것은 불가능하다. 일반적인 운영체제는 기껏해야 밀리초 단위의 정확성으로 제어를 하지만, 통신기기나 정밀 제어의 경우 수십 마이크로초 단위로 정확하게 시간을 측정해야 하는 경우가 많다. 다시 말하자면 인터럽트가 발생하였을 때 해당 프로세스나 태스크가 수십 마이크로초 이내로 동작해야 하는 경우에는 일반적인 범용 운영체제로는 이러한 조건을 만족할 수가 없다. 이러한 점을 만족시키기 위해 실시간 처리가 필요하게 되었으며 실행시간을 만족시켜야 한다[18-20].

VoIP 시스템에서 음성의 녹음과 재생은 그 종료시한을 넘겨 수행을 마쳐도 시스템 운영에는 문제가 없지만, 음성처리의 QoS를 보장하지 못한 음성은 의미가 없어진다. VoIP 시스템의 음성처리(녹음, 재생) 부분의 종료시한은 연성 실시간 처리(soft real-time processing)이지만 VoIP 시스템에서의 음성전달의 지연이나, 끊김 현상은 치명적인 기능상의 문제이므로 음성처리를 위해 경성 실시간(hard real-time) 시스템이 요구된다.

본 논문에서는 리눅스를 기반으로 실시간 음성처리 VoIP 단말기를 설계 구현한다. 설계 구현하는 하드웨어 시스템은 i486 프로세서를 기반으로 설계하며, 음성 코덱칩을 사용하여 실시간으로 음성 데이터를 처리한다. 또한 실시간 음성 데이터를 관리하고 처리하기 위해 리눅스 기반 실시간 처리 운영 체제인 RTLinux를 포팅하여 실시간 음성처리 모듈을 구현한다. RTLinux는 경성 실시간 운영체제를 위해 기존 리눅스에 최소한의 변경으로 선점 가능한 모듈형태로 구현 개발된 실시간 운영체제이며 다양한 API(Application Programming Interface)를 제공한다[18-20]. 음성처리에 사용한 음성처리 모듈은 ITU-T 표준 음성 코덱인 G.723.1 사용하여 30ms 내에 24바이트로 인코딩/디코딩된 음성 데이터를 전송하도록 구현하고, 음성 전달의 QoS를 보장하기 해주기 위해서 리눅스에 실시간 음성처리 디바이스 드라이버를 설계 구현한다. 설계 구현된 시스템의 테스트 및 타당성 검증을 위해 음성채팅 응용 프로그램을 단말기에 구현하여 통화품질을 시험한다.

본 논문의 구성은 다음과 같은 순서로 전개된다. 2장에서 실시간 처리 시스템과 RTLinux를 간략히 소개하고, 3장에서는 설계 개발된 VoIP 단말기 시스템의 하드웨어 구성, 실시간 처리 디바이스 드라이버, 음성채팅 응용 프로그램들을 기술한다. 그리고 4장에서는 통화품질 테스트 환경과 실험 결과를 기술한 후 5장의 결론으로 마친다.

2. 실시간처리 리눅스

2.1 실시간처리 시스템

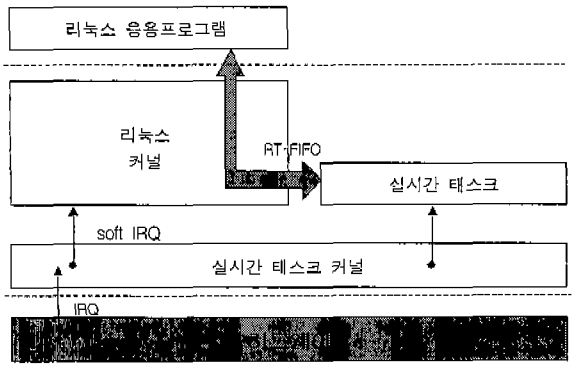
실시간 처리에 대한 정의는 다음과 같다. 실시간 시스템이란 기존의 컴퓨터 시스템과 달리 시스템 동작의 정확성이 논리적 정확성뿐만 아니라 시간적 정확성에서도 좌우되는 시스템을 말한다. 실시간 시스템에 존재하는 시간 제약은 종료시한으로 주어진다. 실시간 시스템이 종료시한을 만족시키기 위해서는 고속의 계산을 요구하게 되지만, 고속의 계산이 실시간 시스템의 요구조건을 만족시키는 것은 아니다. 일반적으로 고속의 계산은 시스템의 평균 응답시간을 최소화하지만, 실시간 시스템에서 요구되는 예측 가능성을 보장하지는 않는다. 예측 가능성이란 시스템의 명세에 정의된 고장이나 작업 부하 조건에서 태스크의 종료시한 만족을 보장하는 것을 의미한다. 최근 운영 체제는 컴퓨터의 성능을 향상시키는데 많은 도움이 되고 있다. 그러나, 실시간 시스템에서의 성능은 단지 평균 실행시간에 의해서만 측정되지는 않는다. 실시간 시스템의 제약조건은 시간 제약 조건 외에도 자원, 우선순위 또는 선행관계, 태스크 간 통신 및 동기화 제약 조건이 있을 수 있다.

실시간 운영체계가 주로 사용되는 분야는 내장 제어 시스템과 같은 특수한 분야이다. 몇 가지 분야를 예로 들면 핵발전소 제어, 산업체 제조 공정 제어, 의료 기구의 모니터링, 항법 제어와 유도, 자동차 엔진 제어, 로봇 제어, 원격 측정 제어 등 수없이 많이 있다. 이들을 크게 두 가지 분류로 나눌 수 있는데 경성 실시간 시스템과 연성 실시간 시스템의 두 분야이다. 이는 시간적 제약 조건이 만족되지 않았을 경우에 야기되는 피해의 정도에 따라 분류된 것인데, 항공기의 제어와 같이 주어진 시간 내에 제어를 하지 못할 경우 인명 피해와 같은 손실이 유발되는 경우를 경성 실시간 시스템, 그렇지 않은 경우에 연성 실시간 시스템이라 한다. eCOS[7], RTAI[5] 그리고 RTLinux[18-20] 등은 리눅스를 수정하여 실시간 시스템의 운영체제로 변형시킨 것으로 기존의 리눅스가 가지는 모든 장점에 실시간 처리 기능을 추가한 형태로, 비상용이고 오픈 소스라는 장점을 가지고 있다.

2.2 RTLinux

리눅스는 일반적인 목적으로 설계된 운영 체제이기 때문에 실시간 처리 운영 체제가 아니며, 실시간 처리 운영 체제의 입장에서 보면 서론에서도 언급했지만 많은 문제를 지니고 있다. 리눅스가 실시간 처리 기능을 갖추려면 실제로 커널부터 다시 설계되어야 하지만, 이는 커널을 구성하는데 많은 시간을 필요로 하며, 리눅스 커널이 업그레이드 될 때마다 이를 신속히 반영하기 어렵고, 버그가 생길 가능성이 더 커지게 된다. 이에 RTLinux는 커널을 다시 설계하

지 않고 리눅스 소스 변경을 최소화하는 방법으로 진행되었다[18, 19]. 리눅스 아래에 RTLinux 커널을 올리고 기존 리눅스 커널을 RTLinux 커널에서 돌아가는 하나의 프로세스로 만들어 실시간 처리 태스크와 기존의 리눅스 커널이 공존하는 형태로 만들었다. RTLinux 커널은 실시간 처리 프로세스를 생성하고 스케쥴링하며, 인터럽트가 발생하였을 때 이를 처리하며 리눅스와 통신을 담당하는 역할을 한다. 기존 리눅스 커널은 그대로 일반 리눅스 프로세스를 관리하고, 자신의 인터럽트를 처리한다.

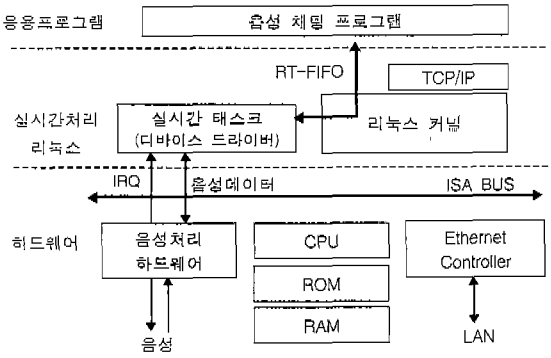


(그림 1) RTLinux 커널 구조

(그림 1)은 RTLinux 커널과 기존의 리눅스 커널과의 관계를 보여준다. 리눅스 커널은 RTLinux 상에서 돌아가는 하나의 태스크로 간주되고, 더 이상 실행할 수 있는 실시간 태스크가 없을 때 실행된다. 기존 리눅스 커널은 선점되지 않는다는 가정 하에 만들어져 있지만 실시간 처리 리눅스에서는 커널 작업중이라도 이를 중단할 수 있어야 하는데, 현실적으로 커널을 재진입 가능하게 수정하는 것은 커널을 다시 설계해야 하는 문제가 있다. 그러나 RTLinux에서 이러한 문제는 실시간 태스크와 실시간 인터럽트가 기존 커널과 무관하기 때문에 기존 리눅스 커널을 선점 하더라도 문제가 되지 않고, RTLinux의 실시간 처리 리눅스 커널 그 자체는 선점할 수 없도록 설계되어있다. 또한 아주 작고 빠르게 설계되어 큰 시간 지연을 발생하지 않는다. RTLinux에서 인터럽트는 기존 리눅스 커널 관리하에 있는 인터럽트와, 실시간 처리 리눅스 커널 관리하에 있는 실시간 인터럽트 두 가지로 나뉜다. 인터럽트가 발생했을 때 실시간 처리 인터럽트 핸들러가 등록되어 있다면 이를 불러주고, 실시간 처리 인터럽트 핸들러가 없거나, 인터럽트를 기존 리눅스 커널과 공유하는 경우 이를 리눅스 커널에 전달한다. RTLinux의 실시간 태스크는 커널 권한을 가지고 수행되므로 자유롭게 하드웨어에 접근할 수 있다. 또한 코드와 데이터 영역 모두 메모리에 존재하며 페이지아웃 되지 않는다. 실시간 태스크에서 할 수 있는 일은 극히 제한되기 때문에, 실제 실시간 기능이 필요한 부분만 실시간 태스크로 구현되고 그렇지 않은 부분은 기존 리눅스 프로세스로 구현된다. RTLinux에서는 실시간 태스크와 기존 리눅스 프로세스와의 데이터 교환을 위하여 RT-FIFO(Real-Time First-In First-Out) 큐를 제공한다. RT-FIFO 큐는 커널 주소공간에 공유 메모리를 이용하여 구현된다. 실시간 태스크에서 생성하고 버퍼의 크기를 고정하기 때문에 사용 중 버퍼가 넘치거나 (overflow), 부족한(underflow) 현상이 발생할 수 있다.

3. 실시간 처리 VoIP 단말기 시스템

(그림 2)는 설계 구현한 VoIP 단말기의 시스템 구성도이다. 코덱칩을 사용하여 음성을 인코딩, 디코딩하는 음성처리 하드웨어를 구현하였고, 하드웨어에서 발생하는 음성 데이터와 인터럽트를 처리하는 디바이스 드라이버를 실시간 태스크 형태로 구현하였다. 또한 음성통신을 할 수 있는 음성 채팅 프로그램을 리눅스에서 구현하였고 실시간 태스크와 음성 채팅 프로그램과의 인코딩된 음성 데이터는 RT-FIFO를 사용하여 통신한다.

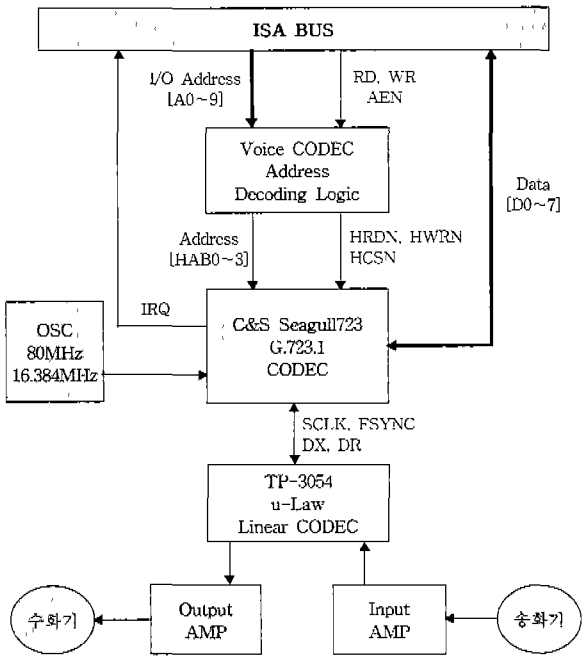


(그림 2) VoIP 단말기 시스템 구성도

3.1 하드웨어 구성

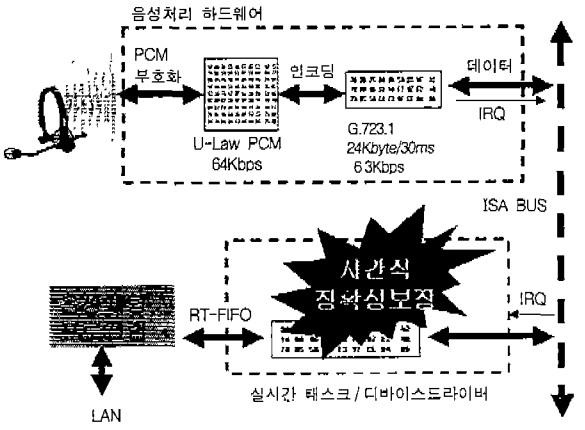
본 논문에서는 G.723.1 음성 코덱을 사용한다. G.723.1은 비트율, 음질, 복잡도, 지연 등의 파라미터들 사이의 최적치를 선택한 음성 코덱으로 6.3kbps와 5.3kbps의 두 가지 비트율을 지원한다[1]. G.723.1 코덱의 전송 속도는 데이터 헤더 부분을 제외하고 6.3Kbps이므로 모뎀(56Kbps)에서도 양방향 통신이 가능하다. 그러나 G.723.1이 작은 대역폭을 보장하지만 호스트 시스템의 프로세스로 처리하기에는 불가능하여 전용 코덱 칩을 사용하여 음성처리 보드를 구현하였다.

(그림 3)은 음성처리 하드웨어의 구성도이다. (그림 3)에서 수화기와 송화기를 통하여 아날로그 음성 신호가 입출력되고, 음성 코덱칩 TP-3054[17]와 Seagull723[16]을 이용하여 아날로그 음성신호를 디지털 데이터로 변환하여 G.723.1으로 인코딩한다. 인코딩된 음성 데이터는 인터럽트 요구 신호와 함께 ISA버스에 전송되도록 설계하였다. 수화



(그림 3) 음성처리 하드웨어 구성도

기로 입력된 아날로그 음성신호는 TP-3054을 통하여 u-Law PCM 64Kbps 신호로 변환된다. 변환된 PCM 신호는 FSYNC, SCLK 같은 동기신호와 함께 선형적으로 코덱칩에 전달되고 코덱칩에서 G.723.1 6.3Kbps 비트율로 변환된다. 그리고 프레임 단위로 음성 데이터를 코덱칩 Rx 버퍼에 저장하고 인터럽트 신호를 발생시킨다. 호스트 시스템에서는 코덱칩의 레지스터를 통하여 Rx 버퍼에 저장된 음성 데이터를 읽어서 처리한다.



(그림 4) 실시간 디바이스 드라이버 동작

3.2 실시간 처리 디바이스 드라이버

본 논문에서 구현한 실시간 처리 시스템은 음성처리 하드웨어의 디바이스 드라이버를 실시간 태스크 형태로 구현하였다. 이러한 모듈은 기본적으로 커널 수준에서 수행되며 경성 실시간 특성을 가진다. (그림 4)는 음성처리 하드웨어의 동작과정과 실시간 태스크, 즉 음성처리 하드웨어의 디바이스 드라이버의 동작을 개념적으로 도식화한 그림이다. 아날로그 음성신호가 음성처리 하드웨어에서 PCM 디지털 음성 데이터로 변환되고 이것을 다시 G.723.1 코덱칩이 프레임 단위로 압축한다. 압축된 음성 데이터는 매 프레임마다 IRQ 신호와 함께 ISA 버스를 통하여 호스트에 전송된다. 이때 호스트 시스템의 실시간 처리 디바이스 드라이버는 한 프레임 간격 안에 현재 요청된 인터럽트를 처리하도록 구현하였다.

G.723.1의 프레임 단위는 30ms이므로 코덱칩에서 30ms 단위로 인코딩된 음성 데이터를 코덱칩 버퍼에 저장하고 인터럽트 요구신호를 발생시킨다. 이때 디바이스 드라이버를 다음 프레임이 버퍼에 저장되기 전에 코덱칩에서 인코딩된 음성데이터를 모두 가져와야 한다. 즉 디바이스 드라이버의 인터럽트 핸들러는 30ms 간격으로 호출되어지고 다음 프레임 인터럽트의 요구가 발생하기 전 60ms 안에 코덱칩의 음성데이터를 모두 처리해야 음성처리의 QoS를 보장할 수 있다.

```
...
void outportb(int port,unsigned char data)
{
        outb_p(data,port) ;             /*메모리 맵 I/O */
}
unsigned char inportb(int port)
{
        return inb_p(port) ;        /*메모리 맵 I/O */
}

int init_module(void)
{                                   /* 모듈의 시작 포인트*/
        ...
                                    /* Real-Time FIFO 생성 */
        rtf_create(CMDF0, BUF_SIZE) ;
        rtf_create(CMDF0, BUF_SIZE) ;
        rtl_request_global_irq(intno, handler)
}

void cleanup_module(void)
{                                   /* 모듈의 끝*/
        ...
         shutdown() ;
}

...
```

```
/* 인터럽트 핸들러, 선점, RT-FIFO사용 */
unsigned int
handler(unsigned int irq_number, struct pt_regs *p)
{    ...
      if(read_mem16(hsr_addr) & RXR){
         /* RT-FIFO로 부터 데이터 읽어옴 */
         res = rtf_get(CMDF1, buf, BUF_SIZE) ;
           for(i = 0 ; i < BUF_SIZE ; i+ = 2){
             /*코덱칩의 Tx 레지스터를 이용 */
             /* Tx-Buffer를 채운다. */
             outportb(tdp_addr, buf[i]) ;
```

```
            outportb(tdp_addr+1, buf[i+1]) ;
        }
        for(i = 0 ; i < BUF_SIZE ; i+ = 2){
        /*코덱칩의 Rx 레지스터를 읽어 옴 */
            buf[i] = inportb(rdp_addr) & 0xff ;
            buf[i + 1]=inportb(rdp_addr + 1)& 0xff ;
        }
        /*RT-FIFO에 압축된 음성 데이터 전달*/
        rtf_put(CMDF0, buf, BUF_SIZE) ;
    }
    ...
    rtl_hard_enable_irq(intno) ;
    return 0 ;
}/* 인터럽트 핸드러 끝 */
```

(그림 5) 실시간 처리 디바이스 드라이버 코드

(그림 5)는 음성처리 하드웨어를 핸들링하는 디바이스 드라이버를 실시간 처리 테스크 형태로 구현한 소스코드의 일부이다. init_module()과 cleanup_module()은 리눅스의 모듈의 시작 포인트와 끝 포인트로, 모듈의 시작부분에서 리눅스 프로세서와의 실시간 태스크간의 데이터 교환 통로인 RT-FIFO를 rtf_create()라는 RTLinux에서 제공하는 API를 이용하여 생성한다. 이렇게 생성된 RT-FIFO는 인터럽트 핸들러에서 rtf_get()과 rtf_pu()를 이용하여 읽거나 쓸 수 있다. 또한 모듈의 시작부분에 rtl_request_global_irq(intno, handler)를 이용하여 실시간 인터럽트의 핸들러를 등록할 수 있다. 이때 intno는 인터럽트 번호이고 handler는 인터럽트가 호출되는 함수이다. 소스코드의 우측 부분이 인터럽트 핸들러의 일부로 인터럽트가 호출되면 상위 for루프에서는 RT-FIFO에서 음성데이터를 읽어서 outportb()를 통하여 코덱칩 Tx 버퍼에 저장하고, 하위 for루프에서는 코덱칩의 버퍼에서 inportb()를 이용하여 코덱칩의 Rx 버퍼에서 인코딩된 음성데이터를 읽어서 RT-FIFO를 통하여 리눅스 프로세서에 전달한다.

3.3 음성채팅 프로그램

```
void PP_RecPlay(void *arg){
    ...
    /* RT-FIFO를 오픈 */
    if((wav_fd0 = open(RTFIFO_REC,O_RDWR)) == -1){
        fprintf(stderr, "Open rtf0 Error\n") ;
        return ;
    }
    ...
    while(iStop == FALSE) {
        res = read(fd_voip,&Msg,lenMsg) ;    /* 메인 프로세스로부
                                                터 메시지 받음*/
        if(res >= lenMsg){
            ...
            if(Msg.iMsg == MSG_START_VOIP){
                            /* UDP 소켓 생성 */
                sprintf(IAddr, "%s", Msg.strMsg) ;
                ...
                sockfd = socket(AF_INET, SOCK_DGRAM, 0) ;
                bind(sockfd, (struct sockaddr *)&my_addr,
                        sizeof(struct sockaddr)) ;
                addr_len = sizeof(struct sockaddr) ;
                ...
                pidRec=fork() ; /* 프로세스 생성 */
                if( pidRec == 0 ){
                    while(1){ /* 코덱칩Rx버퍼 -> RT-FIFO ->
                                Ethernet */
                        res = read(wav_fd0,buf,BUF_SIZE) ;
                        sendto(sockfd, buf, BUF_SIZE, 0
                            ,(struct sockaddr *)&dest_addr
                            , sizeof(struct sockaddr)) ;
                    }
                }
                pidPly = fork() ; /* 프로세스 생성 */
                if( pidPly == 0 ){
                    while(1){ /* Ethernet -> RT-FIFO -> 코덱칩Tx
                                버퍼*/
                        recvfrom(sockfd,buf,BUF_SIZE,0
                            ,(struct sockaddr *)&dest_addr, &addr_len) ;
                        write(wav_fd1,buf,BUF_SIZE) ;
            ...
    ...
}
```

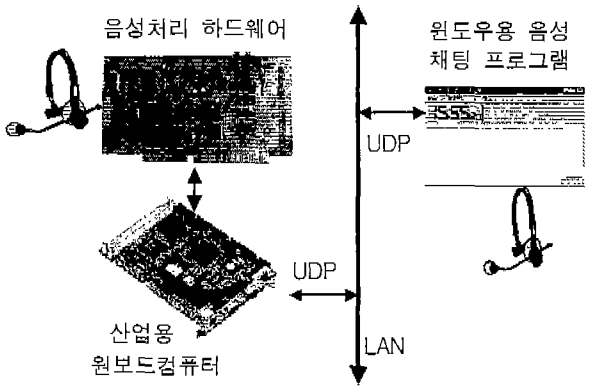
(그림 6) 음성 채팅 프로그램

(그림 6)은 실시간 처리 VoIP 단말기 상에 구현된 음성 채팅 프로그램의 일부이다. 음성을 UDP를 이용하여 전송하는 프로그램은 멀티 프로세스 형태로 구현하였다. 메인 프로세스에서 통화 상대의 IP 주소를 파이프로 받아서 UDP 소켓을 생성하고 코덱칩에서 인코딩된 음성 데이터를 RT-FIFO를 통하여 읽고 sendto()로 전송하는 루틴을 fork()를 이용하여 프로세스로 구현하였다. 같은 방법으로 revform()를 이용하여 UDP소켓을 통하여 읽은 음성 데이터는 write()를 이용하여 RT-FIFO에 쓰여지고 이것은 코덱칩으로 전달되어 음성 신호가 출력되도록 구현하였다. RTLinux에서는 RT-FIFO의 노드는 /dev/rtf0에서 /dev/rtf63까지 64개를 사용할 수 있고 파일 입출력 함수 open(), read() 그리고 write()등을 이용하여 접근할 수 있다.

4. 실험 및 분석

4.1 실험환경

(그림 7)은 VoIP 시스템의 실험환경을 보여주고 있다. 실험에 사용한 하드웨어 환경은 G.723.1 코텍 칩으로 만들어진 음성보드를 RTLinux가 포팅된 i486 산업용 원보드 컴퓨터의 ISA 슬롯에 장착하고 일반 핸드셋과 마이크로폰을 사용하였다. 구현된 음성처리 하드웨어 상에서 본 논문에서 구현한 실시간 처리 모듈을 적재한 후 리눅스용 음성 채팅 프로그램을 실행하고, 상대 컴퓨터는 윈도우에서 구현한 윈도우용 음성채팅 프로그램을 수행시킨다. 음성 채팅 중 여러 작업을 수행시켜 호스트와 클라이언트 시스템에 부하를 주면서 통화 품질을 확인하였다.



(그림7) VoIP시스템 실험 환경

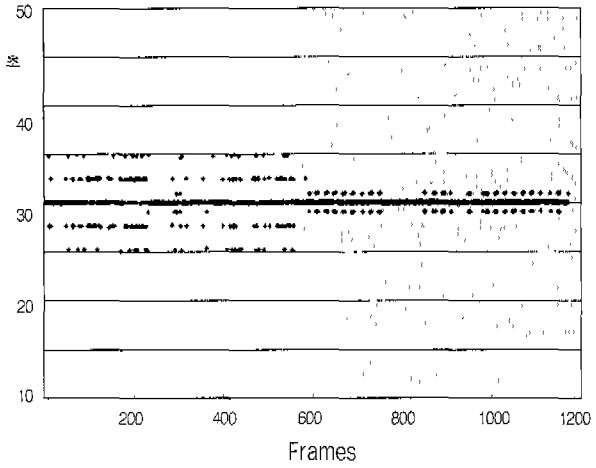
```
unsigned int handler(unsigned int irq_number, struct pt_regs *p)
{
    int i = 0, res ;
    if(irq_number == IRQ5) {
        if((read_mem16(hsr_addr) & RXR)){
                /* 인터럽트 시작 시각... */
            nowTime = gethrtime() ;
            for(i = 0 ; i < WAVE_BUF_SIZE ; i- = 2){
               outportb(tdp_addr,buf[i]) ;
               outportb(tdp_addr+1,buf[i+1]) ;
            }
            for(i = 0 ; i < WAVE_BUF_SIZE ; i- = 2){
               buf[i] = inportb(rdp_addr)& 0xff ;
               buf[i + 1] = inportb(rdp_addr + 1)&0xff ;
            }
                    /*인터럽트 처리후 시각 ..*/
            endTime = gethrtime() ;
            TimeBuf[nTimeBuf++]
                    = nowTime-prvTime ;
            ...
            print_to_k() ; /*시간차 출력*/
            prvTime = nowTime ;
        ...
}
```

(그림 8) 실시간 인터럽트 핸들러 간의 시간측정 소스

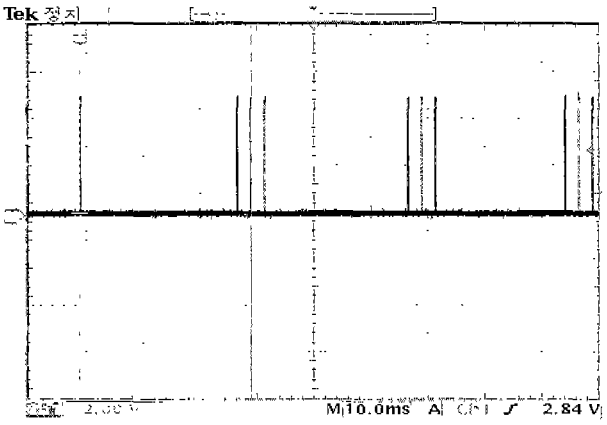
또한 (그림 8)과 같은 소스코드를 사용하여 실시간 인터럽트 핸들러간의 시간차를 측정하였다. 소스 코드에서 nowTime, prvTime를 실시간 디바이스 드라이버내의 전역변수로 선언하고 인터럽트 핸들러가 호출되는 부분에서 RTLinux의 시간측정 함수인 getrtime()를 이용하여 이전 핸들러와 현재 호출된 인터럽트 핸들러간의 시간차를 측정하고 printk()로 출력하여 이를 도식화하였다.

4.2 실험 결과

통화중 음성의 제생과 녹음에는 사람이 감지할 수 있는 끊김 현상은 나타나지 않았고 양호한 통화 품질을 보장해 주었다. 또한 호스트와 클라이언트 시스템에 부하를 주면서 수행시켰을 때의 통화 품질을 확인 한 결과도 마찬가지로 매우 좋은 음성 통화 품질을 보여 주었다.



(그림 9) 인터럽트 처리 시간차



(그림 10) 인터럽트 신호 파형

(그림 9)는 인터럽트의 시간차를 도식화한 그래프로 인터럽트 핸들러의 시간차를 점으로 나타내고 있다. 그림 x축은 프레임을 나타내고 y축은 프레임과 프레임 사이의 시간차를 보여준다. 그래프의 전반부는 높낮이가 복잡한 음성을 마이크로폰으로 입력 할 때이고 후반부는 묵음과 같은 높낮이가 단순한 음성이 입력 될 때이다. 대부분의 점들은 시간 간격이 30ms를 기준으로 분포되어 있지만 이 주기에서 약 ±10ms정도 차를 보이는 점들은 실시간 처리 디바이스의 인터럽트 핸들러의 응답시간의 지연으로 발생하는 것이 아니라 입력 음성의 복잡성에 따라서 코덱칩의 IRQ신호의 주기가 프레임 길이보다 길어지거나 짧아지는데 기인한다. 이와 같은 현상은 (그림 10)과 같이 오실로스코프를 이용하여 인터럽트의 발생 주기를 측정한 결과에 잘 나타나고 있다. 인터럽트 발생 주기를 계측장비를 이용하여 측정한 그래프로 시작 파형을 기준으로 인터럽트가 발생하는 시간차를 하드웨어적으로 측정하였다. 그림에서 나타내듯이 두 번째 이후 파형들의 간격이 30ms를 기준으로 약간 오차를 나타내는 것은 코덱칩이 입력 음성신호의 복잡성에 따라 인터럽트 신호를 30ms를 벗어나거나 모자라게 보낸다는 것을 보여준다.

(그림 8)에서 나타나듯이 인터럽트의 시간차를 나타내는 모든 점들은 프레임 주기인 30ms에 주로 분포되어 있고 다음 프레임 인터럽트가 발생하는 60ms를 초과하는 점들은 나타나지 않아 구현된 VoIP 시스템이 통화품질 뿐 아니라 수치적으로도 실시간 처리로 QoS를 보장할 수 있음을 보여준다.

5. 결 론

본 논문에서는 많은 Real-Time OS 중 일반 리눅스 커널을 기반으로 Real-Time 기능을 커널에 추가하는 RTLinux를 사용하여 VoIP 시스템에서의 Real-Time OS로서 리눅스 사용 가능성과 성능 등을 검토하였다.

설계 구현한 리눅스 기반 실시간 음성처리 VoIP 단말기는 i486 프로세서를 기반으로 하드웨어 시스템을 설계하였고, 음성 코덱칩을 사용하여 실시간으로 음성 데이터를 처리하였으며, 음성 전달의 QoS를 보장하기 해주기 위해서 리눅스 기반 실시간 처리 운영 체제인 RTLinux를 포팅하여 실시간 음성처리 디바이스 드라이버를 설계 구현하였다. 설계 구현된 시스템의 테스트 및 타당성 검증을 위해 LAN 환경에서의 음성채팅 프로그램에 적용하여 통화품질을 시험하고, 인터럽트의 응답성을 측정하기 위하여 핸들러와 핸들러 사이의 시간차를 측정한 결과 좋은 통화 품질과 매우 높은 인터럽트 응답성을 확인하였다.

리눅스는 그 성능과 안정성이 범용 컴퓨터에서 이미 검증되었고 실시간 처리 시스템으로도 포팅이 가능하다. 또한 상용 운영체제보다 가격 경쟁력에서 앞서고, 소스가 공개되어 있어 필요에 따라서 리눅스 커널 자체를 수정하여 사용할 수 있으며 이미 개발된 다양한 응용 프로그램을 사용할 수 있다. 본 논문에서 적용한 시스템과 같이 리눅스는 멀티미디어 통신 장비의 실시간 처리 운영체제로서 충분한 기능과 안정성을 가지고 있어 앞으로 좀 더 광범위한 활용이 기대된다.

참 고 문 헌

[1] 고대식, 박준석, "인터넷 실시간 멀티미디어 통신", 기전연구사.

[2] 여현동, 박원배, "H.323과 SIP의 비교 연구와 VoIP의 발전 전망", 2000년 춘계 정보과학회지.

[3] 이명근, 이상정, 조성범, 임재용, "실시간처리 리눅스 기반 VoIP시스템 설계 및 구현", 정보과학회 2001년 봄학술발표논문집(A), 제28권 제1호, pp.289-251, 200.

[4] 진성근, 서대화 "인터넷폰의 설계 및 구현", 한국정보과학회 '97가을 학술발표논문집(III), pp.523-526, 1997.

[5] 최대수, 임종규, 구용완, "RTLinux에서 효율적인 테스크 스케쥴링을 위한 프레임웍 설계", 2000년 춘계 정보과학회지.

[6] Alessandro Rubini, "Linux Device Drivers," 한빛미디어.

[7] eCOS Manuals, Redhat, http : //www.redhat.com/docs/manuals/ecos/.

[8] Henning Schulzrinne, Jonathan Rosenberg "The IETF Internet Telephony Architecture and Protocols," IEEE Network? May/June, 1999.

[9] ITU-T Recommendation H.323v2, "Packet based Multimedia Communication Systems," 1998.

[10] Ismael Ripoll, "Real Time Linux I, II, III," http : //www.linuxfocus.org.

[11] Jerry Epplin, "Linux as an Embedded Operating System," http : //www.linuxfocus.org.

[12] Matt Welsh, "Implementing Loadable Kernel Modules for Linux," http : //www.ddj.com/ddj/1995/1995.05/welsh.html.

[13] Michael Barabanov and Victor yodaiken, "Real-Time Linux," Linux journal, February, 1997.

[14] R. Magnus, U. Kunitz, M. Dziadzka, D. Verworner M. Beck, H. Bohme, "Linux Kernel Internals," Addison Wesley.

[15] RTAI Programming Guide, Dipartimento di Ingegneria Aerospaziale-Politecnico di Milano Real Time Application Interface, http : //www.aero.polimi.it.

[16] Seagull723 Preliminary Data Book, C&S Technology, http : //www.cnstec.com.

[17] TP3054 Datasheet, National Semiconductor, http : //www.national.com.

[18] Victor Yodaiken, "The RTLinux Manifesto," http : //www.rtlinux.org.

[19] Victor Yodaiken, "The RT· Linux approach to hard real-time," http : //www.rtlinux.org.

[20] Victor Yodaiken, "The RT-Linux approach to hard real-time," http : //www.rtlinux.org.

[21] W. Richard Stevens, "Advanced Programming in the UNIX Environment," McGraw-Hill Companies, Inc.

[22] W. Zhao, K. Ramamritham, John. A. Stankovic, "Preemptive Scheduling Under Time and Resource Constraints," IEEE Computers, Vol. C-36, No.8, pp.949-960, Aug. 1987.

이 명 근

e-mail : lmk77@n-top.com

1999년 순천향대학교 전산학과 졸업(학사)

2001년 순천향대학교 대학원 졸업(석사)

현재 순천향대학교 대학원 과정(박사)

(주)다이알로직코리아 연구개발2팀 팀장

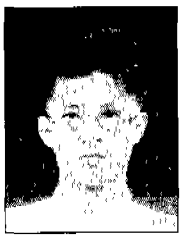
관심분야 : 저전력 소모 프로세서, 임베디드 시스템, VoIP

이 상 정

e-mail : sjlee@sch.ac.kr

1988년~현재 순천향대학교 정보기술공학부 교수

관심분야 : ILP Processor Design, Optimizing Compiler, Embedded Systems, Internet Network for Multimedia Services



서 정 민

e-mail : sjm@kicr.co.kr

1998년 순천향대학교 전산학과 졸업(학사)

2001년 순천향대학교 대학원 졸업(석사)

현재 주)다이알로직코리아 연구개발2팀 재직

관심분야 : 임베디드 리얼타임 리눅스, 임베디드 네크워크 장비

임 재 용

e-mail : contack@kicr.co.kr

1997년 순천향대학교 제어계측공학과 졸업

현재 (주)다이알로직코리아 연구개발2팀 재직

관심분야 : 임베디드 시스템, VoIP Gateway