https://doi.org/10.5392/JKCA.2021.21.01.475

안드로이드 운영체제 상에서 실시간 통신 방법

Method of Real-time Communication in Android OS

우상호, 이상길, 이철훈
충남대학교 컴퓨터공학과

**Sang-Ho Woo(w738495@gmail.com), Sang-Gil Lee(sk0137@cnu.ac.kr),
Cheol-Hoon Lee(clee@cnu.ac.kr)**

요약

안드로이드 운영체제는 스마트폰 시장과 함께 빠르게 자리잡으며 네비게이션, 냉장고 패널 등 다양하게 적용하고 있다. 기존 단말은 정확한 수행 시간이 요구되는 RTOS를 사용하면서 높은 정밀도를 요구하는 서비스의 구현이 가능했으나, 안드로이드는 높은 정밀도의 실시간 성능을 제공할 수 없는 단점이 있다. 본 논문에서는 안드로이드 운영체제에서 동작하는 실시간 통신 방법을 제안한다. 실시간 통신은 UDP 프로토콜을 이용하여 부하를 줄이고, 커널 영역에 고정밀 타이머를 통해서 실시간 운영체제의 타임 틱을 구성한다. 안드로이드 응용 레벨의 라이브러리를 작성하여 실시간 통신을 구현하고 실시간 성능 검증을 위해 기존 안드로이드의 라이브러리와 비교하였다.

■ 중심어 : | 실시간 운영체제 | RTiK | 실시간 시스템 | 임베디드 시스템 | 범용 운영체제 |

Abstract

The Android OS has quickly established itself with the smartphone market and is being applied in a variety of ways such as navigation and refrigerator panels. Existing terminals can implement services that require high precision while using RTOS that requires accurate execution time, but Android OS has a disadvantage in that it cannot provide high-precision real-time performance. In this paper, we propose a real-time communication method that operates in the Android OS. Real-time communication reduces the load using the UDP protocol, and configures the real-time operating system time tick through a high-precision timer in the kernel area. An Android application level library was created to implement real-time communication and compared with the existing Android library for real-time performance verification.

■ keyword : | RTOS | RTiK | Real-Time System | Embedded System | GPOS |

I. 서 론

최근 스마트가전, 헬스케어, 웨어러블기기 등 여러 분야에서 IoT(Internet Of Things)와의 융합이 이루어짐에 따라 IoT 기반의 디바이스의 수가 증가하고 있다[1-4]. 또한 IoT에 적용되는 RTOS(Real-Time Operating System)은 IoT 기반 디바이스에서 발생하는 전력 소모 문제, 메모리 보호 기능 문제와 같은 복잡

* 이 연구는 충남대학교 연구장려장학금에 의해 지원되었음
접수일자 : 2020년 11월 27일
수정일자 : 2021년 01월 19일
심사완료일 : 2021년 01월 19일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

한 문제를 해결하는데 도움을 주고 있다[5-7]. 이러한 주변 기기를 통합하기 위해 안드로이드가 탑재된 스마트폰을 사용하고 있으며, 다양한 라이브러리와 GUI를 바탕으로 스마트폰 및 다양한 기기에서 사용되고 있다.

안드로이드가 탑재된 다양한 기기가 사용되면서, 이를 보다 전문적인 용도의 사용이 고려되고 있다.

군용 점검장비는 미사일 등의 기능 점검을 위해 사용되는 장비로 데이터 획득이나 평가에 실시간 성능이 매우 중요하게 요구되고 있으며, 기존 x86 기반의 윈도우가 탑재된 시스템을 사용하였다[8-10].

기존의 상용 실시간 운영체제가 사용된 점검 장비휴대성을 높이고자 지속적으로 단말의 소형화를 진행하고 있지만, 상용 실시간 운영체제가 사용된 단말은 기기의 성능을 보장 받기 위해 높은 구매 비용과 별도의 단말 구매 비용이 요구된다. 최근 하드웨어의 발달로 인하여 스마트폰과 같은 모바일 단말의 성능이 랩탑을 따라오고 있어, RTOS를 사용할 수 있는 환경이 만들어지고 있다[10]. 모바일 단말을 RTOS가 적용된 군용 점검장비로 사용하게 된다면, 기존 군용 점검장비의 문제점인 휴대성 문제와 라이선스 문제를 해결할 수 있게 된다.

모바일 단말에서는 안드로이드 플랫폼이 가장 높은 점유율을 차지하고 있으나[13], 현재 상용 안드로이드 플랫폼은 안드로이드 동작에 기반이 되는 리눅스 커널에서 사용하고 있는 태스크 스케줄링 방식은 태스크 간 공평한 자원 사용을 위한 스케줄링을 제공하기 때문에 실시간 성능 제공에 적합하지 않는다는 문제점이 있다. 또한 안드로이드의 응용 프로그램은 커널 바로 위에서 동작하지 않고 자바 가상 머신 위에서 동작하여 실시간 성능을 지원하기 힘들다는 문제점을 지니고 있다[14][15].

본 논문에서는 안드로이드에 실시간 성능을 제공하기 위하여 이중 커널 구조를 구현하여 안드로이드에 실시간 성능을 지원하고, RTOS가 적용된 IoT를 위한 점검 장비로 사용하기 위하여 실시간 통신 방법을 설계 및 구현하였다.

본 논문의 구성은 2장에서 관련 연구를 설명하고, 3장에서 안드로이드 실시간 통신 지원을 위한 리눅스 커널 구조 및 실시간 통신 구현 방법에 대해 설명한다. 4장에서 실험 방법 및 결과를 제시하고, 5장에서는 결론 및 향후 연구를 기술한다.

II. 관련 연구

1. RTAndroid

RTAndroid(Real-Time Extension for Android)는 독일 RWTH Aachen University에서 제안한 실시간 성능을 제공하는 솔루션이다. RTAndroid는 Linux 커널에 패치를 적용하여 선점 가능한 커널로 변경하였으며, Dalvik VM를 수정하였다. 또한 실시간 가비지 컬렉션 알고리즘인 슬랙 기반 스케줄링 방식으로 수정하여 연성 실시간 성능을 제공할 수 있도록 하였다[16].

RTAndroid는 Dalvik VM을 수정하는 등 내부 구조를 변경하기 때문에 각 기기마다 다르게 패치해야 하며, 라즈베리파이3, odroid xu3 및 일부 삼성 안드로이드 스마트폰을 대상으로 적용되었으나, 본 논문에서 실행할 퀄컴 스냅드래곤 820 칩셋이 사용된 기기에서 실행이 불가능하다[20].

2. RTDroid

RTDroid는 기존 안드로이드 스택을 수정하여 안드로이드 어플리케이션에 실시간 성능을 제공하는 솔루션이다. RTDroid는 하드 리얼타임 환경을 지원하는 자바 가상 머신인 Fiji VM을 추가하였으며, 실시간 성능을 보장할 수 있도록 RTDroid 프레임워크를 추가하였다. RTDroid에서 비 실시간 태스크는 기존 안드로이드 런타임인 Dalvik위에서 동작하게 되며, 실시간 태스크는 새로 추가된 Fiji VM위에서 동작하게 하여 기존 안드로이드 응용 프로그램에 대한 호환성을 지원할 수 있게 하였다[17].

RTDroid 또한, 기기 호환성을 위해서 커널 패치가 이루어져야 하는 부분으로 인해 퀄컴 스냅드래곤 820 칩셋이 사용된 기기에서 실행이 불가능한 문제가 있다.

3. RTiKA

RTiKA(Real-Time implant Kernel for ARMLinux)는 ARM 프로세서 기반의 리눅스 환경에서

정확한 주기에 따른 실시간 성능을 지원하기 위해 설계 및 구현되었다[18].

그러나 RTiKA는 리눅스 환경에서 실행이 가능하기 때문에 이를 안드로이드에서 그대로 적용하는 것은 불가능하기 때문에, ARM 프로세서가 적용된 안드로이드 환경에서 실행하기 위해서는 확장 구현을 해야 하는 단점이 있다.

Ⅲ. 안드로이드에 실시간 통신 구현을 위한 RTiK-Android 설계

본 장에서는 안드로이드 환경에서 실시간 성능 제공을 위하여 RTiKA를 개선하여, 안드로이드에 적용한 RTiK-Android의 구조와 동작 과정을 설명하고, 안드로이드 어플리케이션에서 RTiK-Android를 제어할 수 있는 과정에 대해 설명한다.

1. RTiK-Android의 실시간 성능 제공을 위한 구조

RTiK-Android의 실시간 성능을 제공하기 위한 스택 구조는 [그림 1]과 같다.

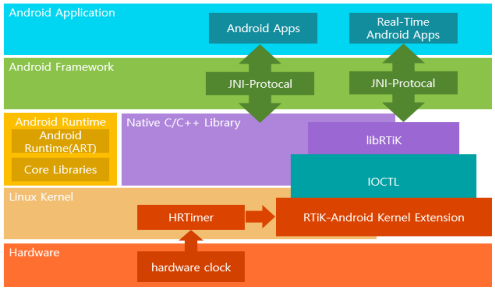

그림 1. RTiK-Android 스택 구조

안드로이드에 실시간 성능을 제공하기 위하여 안드로이드 스택 하단에 존재하는 Linux 커널 영역에 이중 커널인 RTiK-Android Kernel Extension을 구현하였다.

libRTiK은 실시간 안드로이드 응용 프로그램에서 RTiK-Android를 제어하고, 실시간 성능을 제공 받기 위해 만들어진 라이브러리이다. libRTiK은 RTiK-Android Kernel Extension으로 접근하기 위하여 IOCTL을 사용한다.

실시간 안드로이드 응용 프로그램은 실시간 태스크를 생성하기 위하여 RTiK-Android에서 제어되는 실시간 쓰레드를 생성하며, 생성된 실시간 쓰레드를 통하여 안드로이드 응용 프로그램에 실시간 성능을 제공한다.

실시간 성능을 보장하기 위해 사용하는 타이머는 리눅스 커널에서 지원하는 소프트웨어 타이머인 HRTimer를 사용하였으며, HRTimer를 통하여 TSC, HPET, PIT 중 하나를 하드웨어 타이머로 사용한다[19].

2. RTiK-Android 동작 과정

RTiK-Android에서의 실시간 안드로이드 응용 프로그램의 동작 과정은 [그림 2]와 같다.

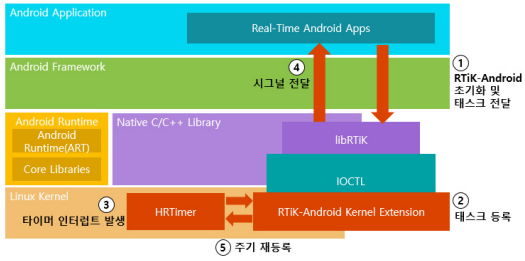

그림 2. 실시간 안드로이드 응용 프로그램 동작 과정

실시간 안드로이드 응용 프로그램은 가장 먼저 RTiK-Android Kernel Extenstion을 초기화 한 뒤 실시간 태스크를 JNI-Protocal을 통하여 RTiK-Android Kernel Extension으로 전달한다. 전달된 태스크는 RTiK-Android의 제어 인터페이스를 통해 실시간 쓰레드를 생성한 뒤, RTiK-Android Kernel Extension에 등록한다.

HRTimer에서 발생한 타이머 인터럽트를 통해 RTiK-Android 스케줄러를 동작시키며, 스케줄러는 등록된 실시간 쓰레드 중 주기에 맞는 태스크에 시그널을 보내어 동작시킨다. HRTimer는 one-shot모드로 동작하기 때문에 RTiK-Android 스케줄러는 다음 HRTimer가 동작할 주기를 재등록한다.

3. RTiK-Android Kernel Extension

RTiK-Android Kernel Extension은 리눅스 커널에서 제공하는 고성능 실시간 타이머인 HRTimer의 제어와 실시간 성능을 요구하는 사용자 영역의 태스크를 관리하는 역할을 한다[19].

3.1. 실시간 타이머 제어

RTiK-Android Kernel Extension은 HRTimer를 통하여 실시간 성능을 지원할 수 있도록 하였다. RTiK-Android Kenrel Extension에서 HRTimer의 제어를 위해 구현된 함수는 [표 1]과 같다.

표 1. HRTimer 제어를 위해 구현된 함수

함수명	설명
rtik_systimer_start_request	HRTimer을 동작시키는 함수
callback_hrtimer	HRTimer의 동작을 처리하는 함수
rtik_systimer_stop	HRTimer의 동작을 중지하는 함수

rtik_systimer_start_request 함수는 RTiK-Android로부터 HRTimer의 동작 활성화를 요청할 때 호출되는 함수이다.

```
int rtik_systimer_start_request()
{
    unsigned long delay_in_ms = 5;

    ktime = ktime_set( 0, delay_in_ms * 1E6L );
    hrtimer_init( &hr_timer, CLOCK_MONOTONIC, HRTIMER_MODE_REL );
    hr_timer.function = &callback_hrtimer;

    isTimerInit = 1;
    hrtimer_start ( &hr_timer, ktime, HRTIMER_MODE_REL );

    return 0;
}
```

그림 3. HRTimer 동작 활성화를 위한 코드

RTiK-Android로부터 rtik_systimer_start_request 함수 호출시, 미리 설정한 주기를 ktime에 등록하며, HRTimer를 MONOTONIC clock 타입으로 초기화한다. 그 뒤 HRTimer의 타이머 인터럽트가 발생할 때 동작할 callback 함수를 HRTimer 인터럽트 핸들러에 등록하며, HRTimer을 요청한 ktime 만큼의 상대 시간 후에 동작하도록 요청한다.

callback_hrtimer 함수는 HRTimer가 요청한 ktime 만큼의 상대 시간 후에 실행될 동작을 구현한 함수이며 [그림 4]와 같다.

```
enum hrtimer_restart callback_hrtimer( struct hrtimer * timer )
{
    ...

    currtime = ktime_get();

    if( timer_func_count > 0 ) {
        while (index < timer_func_count) {
            func = timer_func_list[index];
            if(func!=NULL)func();
            index++;
        }
    } else {
        isTimerEnable = 0;
        return HRTIMER_NORESTART;
    }

    ...

    if ( isRequestStopTimer ) {
        ...
        return HRTIMER_NORESTART;
    }
    hrtimer_forward(timer, currtime, ktime);
    return HRTIMER_RESTART;
}
```

그림 4. HRTimer 동작 시 실행할 함수

callback_hrtimer 함수가 호출되면, 현재 시간을 ktime의 형태로 획득하며, timer_func_list에 등록된 함수를 호출하여 주기적으로 timer_func_list에 등록된 함수들이 작동할 수 있도록 한다. 만약 HRTimer의 중지를 요청받았을 경우 HRTIMER_NORESTART를 반환하여 HRTimer의 동작을 중지하며, 그렇지 않았을 경우에는 HRTimer의 만료 시각을 callback_hrtimer 함수가 호출된 시간으로부터 ktime만큼 연장시켜 HRTimer의 동작이 유지될 수 있도록 한다.

rtik_systimer_stop 함수는 RTiK-Android로부터 HRTimer의 동작 중지를 요청받았을 때 호출되는 함수이며 [그림 5]와 같다.

```
int rtik_systimer_stop()
{

    if (isTimerEnable==0)return 0;
    isRequestStopTimer = 1;
    printk("HR Timer Stop Request ! %d \n", isRequestStopTimer);

    return 0;

}
```

그림 5. HRTimer 동작 중지를 위한 함수

rtik_systimer_stop 함수가 호출되면 isTimerEnable변수를 통하여 현재 HRTimer의 동작 여부를 확인한다. HRTimer가 동작 중인 경우 isRequestStopTimer 변수를 1로 세팅하여 callback_hrtimer함수가 호출 되었을 때 HRTimer의 만료 시각을 연장시키지 않게 함으로 HRTimer의 동작을 중지시킨다.

3.2. 실시간 성능을 요구하는 사용자 영역의 태스크 관리 방법

RTiK-Android Kernel Extension은 실시간 성능을 요구하는 사용자 영역의 태스크를 관리하기 위하여 먼저 사용자 영역에서 작성 된 태스크를 RTiK-Android Kernel Extension으로 전달한다. 전달받은 태스크는 [그림 6]과 같은 RTiK_ProcInfo_t 구조체의 리스트를 통해 관리된다.

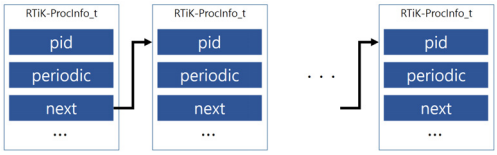

그림 6. 실시간 태스크 정보 리스트의 구조

RTiK-Android Kernel Extension에 등록된 태스크는 실시간 쓰레드를 생성한 뒤 RTiK-Android 스케줄러의 호출을 기다리게 된다.

RTiK-Android 스케줄러는 HRTimer에서 발생한 타이머 인터럽트를 받으면 현재 관리 중인 태스크의 주기를 검사하여 해당 태스크가 동작해야 하는지 검사하게 된다. 동작해야 하는 태스크가 존재할 시 스케줄러는 해당 태스크에게 시그널을 전송하여 대기중인 태스크를 깨우게 된다.

등록된 실시간 태스크의 주기를 관리하기 위하여 RTiK_Global_periodic_count 변수와 RTiK_Global_hyper_periodic 변수를 사용한다. RTiK_Global_periodic_count 변수는 스케줄링 함수가 동작할 때마다 값이 증가되어, 등록된 태스크의 호출 주기를 판단하기 위해 사용된다. 이때 RTiK_Global_periodic_count가 무한히 증가하게 되면 오버플로우가 발생할 수 있다는 문제가 존재한다. 오버플로우 문제를 해결하기 위하여 RTiK_Global_hyper_periodic 변수에 등록된 모든 실시간 태스크 주기의 최소공배수를 구하여 저장한다. 스케줄러가 동작하여 RTiK_Global_periodic_count 변수의 값과 RTiK_Global_hyper_periodic 변수의 값이 같아지면 RTiK_Global_periodic_count 의 값을 0으로 초기화하여 오버플로우가 발생할 수 있는 상황을 방지하였다.

4. RTiK-Android를 이용한 실시간 통신

4.1 실시간 통신 구조

실시간 통신을 구현하기 위해선 빠른 주기로 실행되는 통신 태스크가 통신 시간에 의해 데드라인을 벗어나지 않도록 해야 한다. 그러므로 실시간 통신을 위해서는 신뢰성을 보장하지만, 속도가 느린 TCP 통신을 사용하지 않고 신뢰성이 보장되지 않지만, 속도가 빠른 UDP통신을 사용하여 실시간 통신을 구현해야 한다.

[그림 7]은 윈도우 서버와 안드로이드 클라이언트 사이의 UDP를 사용한 실시간 통신 구조에 대해 나타낸 그림이다.

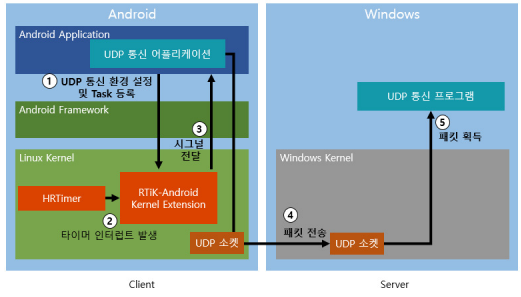

그림 7. 안드로이드 Client와 윈도우 Server와의 UDP통신 과정

실시간 통신을 하기 위해서 Client의 안드로이드 어플리케이션에서 UDP 통신 어플리케이션을 구현한다. 그 뒤 구현된 UDP 통신 어플리케이션은 서버와의 연결을 위해 환경 설정을 한 뒤 RTiK-Android Kernel Extension에 통신 태스크를 등록한다. RTiK-Android 스케줄러는 HRTimer로부터 발생한 타이머 인터럽트를 받게 되면 등록된 실시간 통신 태스크를 호출하고, 호출된 UDP 통신 어플리케이션은 서버에 존재하는 UDP 소켓을 통해 데이터를 전송하여 실시간 통신을 수행한다.

4.2. 실시간 통신 구현

RTiK-Android에서 실시간 UDP 통신 환경 설정을 위해 JNI에서 작성된 sendSocketConnect 함수를 [그림 8]과 같이 구현하였다.

```
extern "C"
JNIEXPORT long
JNICALL
Java_rtik_RTiK_lapi_ljni__1sendSocketConnect(
        JNIEnv *env,
        jobject /* this */
) {
    char recv_ch[100];
    char send_ch[25] = "is receive?";
    time_t timer;
    timer = time(NULL);
    struct tm * t;

    time(&timer);
    t = localtime(&timer);

    server_addr_size = sizeof(server_addr);
    sock = socket(PF_INET, SOCK_DGRAM, 0);
    if( sock == -1) {
        ...
    }

    memset(&server_addr, 0, server_addr_size);
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr(IP);

    sendto(sock, send_ch, sizeof(send_ch), 0,
        (struct sockaddr *)&server_addr, server_addr_size);

    return recvfrom(sock, recv_ch, sizeof(recv_ch), 0,
            (struct sockaddr *)&server_addr, &server_addr_size);
}
```

그림 8. 서버와의 연결 확인을 위한 코드

sendSocketConnect 함수가 호출 시, 먼저 서버에 보낼 데이터를 생성하고 UDP 소켓을 작성한다. Native C/C++ 라이브러리에서 지원하는 socket 함수를 사용하여 IPv4 인터넷 프로토콜을 사용하는 UDP/IP 소켓을 생성하고 지정된 포트와 연결하게 된다. 서버의 포트와 정상적으로 연결 여부를 확인하기 위하여 데이터를 전송한 뒤 서버로부터 전달받은 패킷을 확인한 뒤 종료된다.

5. JNI를 통한 RTiK-Android 제어 방법

안드로이드의 사용자 영역은 자바 응용 프로그램을 통해 동작한다. 이러한 자바 응용 프로그램은 실제 하드웨어 위에서 동작하는 것이 아니라 자바 가상 머신 위에서 동작하기 때문에, 리눅스 커널에서 제공하는 기능을 직접 호출할 수 없다. 그러므로 기존 안드로이드에서 제공하는 JNI(Java Native Interface)를 통하여 C++언어로 작성되어 커널에 접근할 수 있는 인터페이스인 libRTiK에 접근할 수 있도록 하였다.

JNI는 안드로이드 스택에서 안드로이드 어플리케이션과 Native C/C++ 라이브러리 영역에 접근법을 제공한다. JNI를 통한 안드로이드 어플리케이션 영역과 C/C++ 라이브러리 영역의 동작은 [그림 9]과 같다.

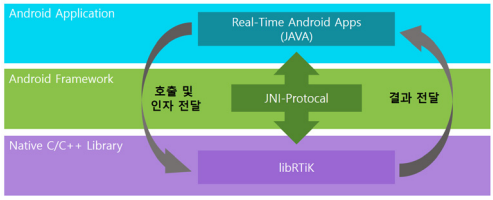

그림 9. JNI-Protocol을 통한 데이터 전달

RTiK-Android를 사용하기 위하여 작성된 JNI는 [그림 10]와 같은 제어 인터페이스 구조를 가진다.

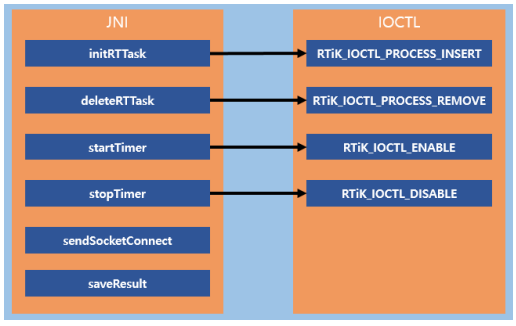

그림 10. RTiK-Android에서 구현된 JNI-Protocol의 구조

실시간 UDP통신을 하면 손실되는 패킷이 존재하기 때문에 정확한 데이터의 측정이 되지 않을 수 있다. 안드로이드 디바이스에서 통신 태스크의 발생 주기를 기록하기 위해 실시간 통신이 끝난 뒤 발생한 통신 데이터를 파일에 저장하는 saveResult 함수를 통해 UDP 통신에서 손실되는 패킷이 여부와 손실된 데이터의 파악이 가능하도록 하였다.

Ⅳ. 실험 환경 및 실험 결과

본 장에서는 RTiK-Android Kernel Extension에서 사용하는 타이머의 주기 보장 여부와 실시간 태스크의 실시간 성능 보장 여부를 확인하기 위하여 설정된 주기와 실제 동작 주기를 비교한다. 또한 RTiK-Android 사용 유무에 따른 비실시간 태스크와 실시간 태스크의 작동 주기를 비교한다.

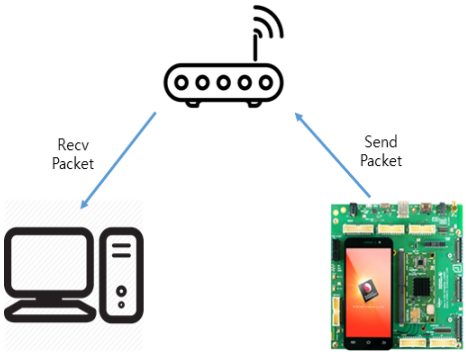

그림 11. 실시간 통신 성능 측정 환경

1. 실험 환경 및 방법

[표 2]은 RTiK-Andorid의 설정된 주기와 실제 동작 주기를 비교하기 위한 환경을 나타내는 표이다.

표 2. Target Board의 하드웨어 환경

TARGET BOARD	
Device	Intrinsys사의 Open-Q820
OS	Android 6.0 Inplemented RTiK-Android
CPU	Qualcomm® Kryo™ CPU, Quad-Core, 64-bit, 2.2GHz

Qualcomm® Kryo™ CPU, Quad-Core, 64-bit, 2.2GHz는 big.LITTLE 솔루션이 적용된 CPU이기 때문에 실시간 태스크가 LITTLE코어로 작동할 수 있다는 문제점이 존재한다. 실험을 진행한 Target Board에서는 LITTLE코어의 작동을 중지하여 Big코어만 동작하도록 하였다.

RTiK-Android의 사용 유무에 따른 통신 주기를 비교 분석하기 위한 실험 환경은 [그림 11]과 같다. Target Board는 Host PC로 Wi-Fi 공유기를 통한 내부망을 사용하여 실시간 통신 태스크가 동작한 시간이 기록된 패킷을 전송한다.

표 3. Target Board 및 Server의 환경

분류	TARGET BOARD	SERVER
Device	Intrinsys사의 Open-Q820	PC
OS	Android 6.0 Implemented RTiK-Android	Windows 10
CPU	Qualcomm® Kryo™ CPU, Quad-Core, 64-bit, 2.2GHz	Intel Pentium CPU G4600 @ 3.60GHz

[표 3]는 Host PC와 TARGET BOARD의 하드웨어 및 소프트웨어 환경을 나타낸 표이다.

실시간 태스크의 유효 주기는 설정된 주기의 두 배의 시간으로 정의되며, 유효 주기를 측정하는 공식은 “태스크의 현재 호출 시간 - 태스크의 이전 호출 시간” 이다.

본 논문에서 적용한 RTiK-Android를 통해서 기존 RTAndroid나 RTDroid와 성능 비교가 단일 플랫폼에서 불가능하기 때문에, TARGET BOARD에 기본으로 적용되어 있는 안드로이드 라이브러리 중, usleep() 함수를 통해서 비교하도록 한다. 기존 Real-Time Android 계열과 직접적인 비교는 어렵지만, 태스크의 주기성이 만족되는 것을 비교하고, 이를 통해서 실시간 통신의 성능이 제공되는 것을 확인한다[21].

2. 실험 결과

2.1. 실시간 타이머 주기 테스트 결과

RTiK-Android를 통한 실시간 태스크의 성능을 분석하기 위하여 10ms 주기를 가지는 실시간 태스크의 10만번 측정 실험 결과는 [그림 12]과 같이 나타났으며, [표 4]는 실험 결과를 분석한 표이다.

표 4. RTiK-Android에서 10ms 주기를 가지는 실시간 태스크의 측정 결과 분석

구분	측정 값	구분	측정 값
전체 데이터	100,000개	최대값	16.7687ms
오차 10% 이상	705개	최소값	2.2496ms
오차 50% 이상	6개	평균값	10.0000ms

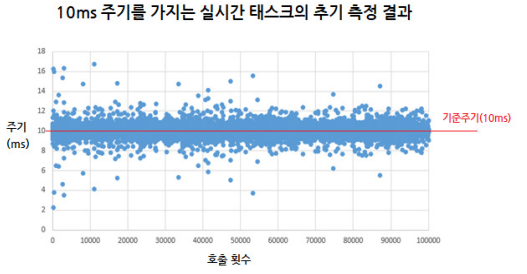

그림 12. 10ms 주기를 가지는 실시간 태스크의 측정 결과

안드로이드 응용 프로그램은 자바 가상 머신 위에서 동작하기 때문에 HRTimer에서 발생한 타이머 인터럽트의 주기 보다 큰 지터를 보인다. 그럼에도 설정된 주

기의 2배인 20ms을 넘는 데이터가 존재하지 않음으로 보아 RTiK-Android의 실시간 성능이 보장됨을 알 수 있다. 또한 설정된 주기의 10%인 9ms에서 11ms사이의 주기를 가지는 데이터가 99.30%를 가지는 곳으로 보아 대부분의 태스크가 설정된 주기에 맞게 동작하는 것을 알 수 있다.

2.2 실시간, 비실시간 통신 태스크간의 성능 비교

RTiK-Android를 사용하지 않고 usleep() 함수를 통해 주기적인 통신을 진행하는 통신 태스크와 RTiK-Android를 통한 실시간 통신 태스크를 10ms 주기로 10분간 측정한 실험 결과는 [그림 13]과 [그림 14]로 나타내었으며, [표 5]는 실험 결과를 분석하여 나타내었다.

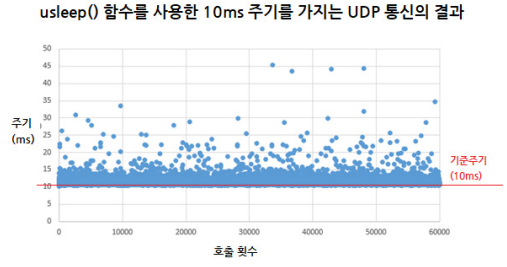

그림 13. usleep() 함수를 이용한 10ms 주기의 UDP 통신 측정 결과

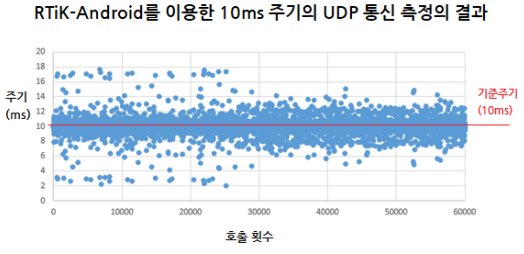

그림 14. RTiK-Android를 이용한 10ms 주기의 UDP 통신 측정 결과

표 5. usleep() 함수와 RTiK-Android를 이용한 10ms 주기의 UDP 통신 측정 결과 비교

구분	측정 값	
	usleep()	RTiK-Android
전체 데이터	53,863	**60,000**
주기 벗어남	55	**0**
오차 10%이상	24,106	**1,136**
오차 50%이상	207	**64**
표준 편차	0.752676	0.401778
최대값	45.3246 ms	17.7101 ms
최소값	10.2664 ms	2.0378 ms
평균값	11.1438 ms	10.0001 ms

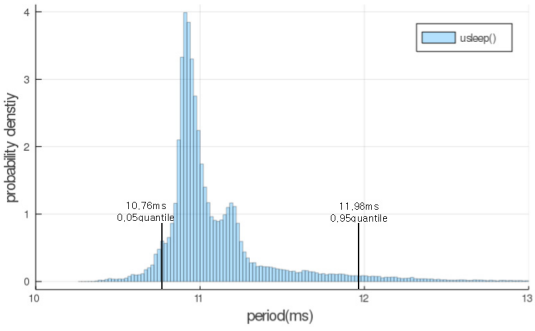

그림 15. usleep() 함수를 이용한 10ms 주기의 UDP 통신의 확률 밀도 함수

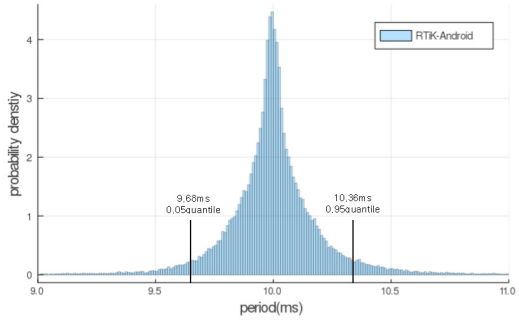

그림 16. RTiK-Android를 이용한 10ms 주기의 UDP 통신의 확률 밀도 함수

RTiK-Android를 사용하지 않고 usleep() 함수를 통해 주기적인 통신을 진행하는 통신 태스크는 10분간의 측정 시간 중 53,893번의 기록이 진행되었다. 이는 60,000번 동작하는 태스크가 주기를 6,107번 놓쳐 정상적으로 동작하지 못함을 나타낸다.

또한, 설정된 주기의 10% 오차를 가지는 데이터는 55.25%에 불가하였으며, 주기를 완전히 벗어나는 데이터 또한 측정되었다.

RTiK-Android를 통한 10ms 주기의 실시간 통신 태스크는 10분간 측정 시간 중 60,000번 동작하여 데드라인을 만족시키지 못한 동작이 존재하지 않음을 나타낸다. 또한, 설정된 주기의 10% 오차를 가지는 데이터가 98.11%를 가지는 곳으로 보아 통신으로 인해 실시간 성능이 떨어짐을 나타내지만, 90% 이상의 태스크가 4% 오차율 이내에서 동작하는 것을 확인할 수 있다.

V. 결론 및 향후 연구 방향

최근 IoT 기반의 디바이스의 수가 증가하고 있으며, IoT 기반 디바이스에서 발생하는 문제를 해결하는데 RTOS가 도움을 주고 있다. 또한 이를 활용하는 모바일 단말의 활용이 높아지면서, 기존의 VxWorks와 같은 상용 실시간 운영체제가 사용된 군용 무기점검 장비를 대체할 수 있도록 휴대용 단말에서 실시간 성능이 요구된다.

따라서 본 논문에서는 안드로이드에 실시간 성능을 제공하여 기존 점검 장비의 문제점을 해결하고자 하였다. 가상 머신을 사용하는 안드로이드 플랫폼의 문제점을 해결하기 위하여 실시간 성능 지원이 필요한 안드로이드 응용 프로그램을 관리하기 위한 구조를 이중 커널인 RTiK-Android Kernel Extension을 구현하였으며, 리눅스 커널에서 지원하는 HRTimer를 사용하여 실시간 성능을 제공하였다. 안드로이드 응용 프로그램에서 RTiK-Android Kernel Extension을 사용할 수 있도록 JNI-Protocal을 사용한 인터페이스를 적용한 RTiK-Android를 구현하였다. RTiK-Android를 군용 점검장비에 사용하기 위한 요구 사항을 만족하기 위해 실시간 통신 프로그램을 구현하였다.

RTiK-Android의 성능을 측정하기 위하여 HRTimer의 인터럽트 발생 주기를 측정하였으며, 안드로이드 응용 프로그램의 동작 주기와 통신 프로그램의 동작 주기를 측정하였다. 리눅스 커널에서 발생하는 HRTimer의 인터럽트 지터에 비해 안드로이드 응용 프로그램의 지터가 매우 큰 것을 확인할 수 있었지만, 안드로이드 응용 프로그램에서 10ms 주기에 맞추어 실시간 통신이 가능한 것을 확인하였다.

향후 연구로는 안드로이드 응용 프로그램에서 발생하는 지터를 HRTimer 인터럽트가 발생하는 지터와 같을 수 있도록 RTiK-Android의 성능을 개선하는 연구와 RTiK-Android 외의 실시간 안드로이드 OS와의 성능 비교 평가를 수행하는 연구가 필요하다.

참 고 문 헌

[1] http://www.epnc.co.kr/news/articleView.html?idxno=47042, 2020.11.27.

[2] https://estimastory.com/2011/08/20/andreessen/, 2020.11.27.

[3] KT Smart Home, https://product.kt.com/wDic/index.do?CateCode=6018, 2020.11.27.

[4] LG Smart Home, https://social.lge.co.kr/tag/%EC%8A%A4%EB%A7%88%ED%8A%B8%ED%99%88/, 2020.11.27.

[5] 최재훈, Barde Stephane Remy Antoine, 김주현, “헬스케어와 사물인터넷 융합기술 동향,” 한국통신학회지(정보와통신), 제31권, 제12호, pp.10-16, 2014.

[6] 최영재, “IoT 장치의 설계 문제 해결과 완전한 기능의 RTOS 사용에 따른 이점,” 2015. http://www.epnc.co.kr/news/articleView.html?idxno=48681

[7] 임은혜, 김영천, “사물인터넷(IoT) 기반 풍력발전기 실시간 모니터링 시스템 구현,” 한국통신학회 추계종합학술발표회 논문집, pp.123-124, 2015(11).

[8] 주민규, 이진욱, 김종진, 조한무, 박영수, 이철훈, “x86 기반의 윈도우즈 상에서 실시간성 지원 방법,” 한국차세대컴퓨팅학회 논문지, 제7권, 제4호, pp.47-58, 2011.

[9] 조아라, 송창인, 이철훈, “윈도우즈 상에서 실시간 디바이스 드라이버를 위한 통합 미들웨어,” 한국콘텐츠학회논문지, 제13권, 제3호, pp.22-31, 2013.

[10] 박지윤, 조아라, 김효중, 최정현, 허용관, 조한무, 이철훈, “태블릿 PC 환경의 실시간 처리 기능 지원,” 한국콘텐츠학회논문지, 제13권, 제11호, pp.541-550, 2013.

[11] 김주만, 송창인, 이철훈, “리눅스용 실시간 이식 커널의 설계,” 한국콘텐츠학회논문지, 제11권, 제9호, pp.45-53, 2011.

[12] 이상길, 이승율, 이철훈, "리눅스 사용자 영역에 실시간성 제공을 위한 미들웨어," 한국콘텐츠학회논문지, 제16권 제5호, pp.217-228, 2016.
[11] Z. He, A. Mok, and C. Peng, "Timed RTOS Modeling for Embedded System Design," Real Time and Embedded Technology and Applications Symposium(RTAS), 2005.
[12] 박병률, 맹지찬, 이종범, 유민수, 안현식, 정구민, "RTOS기반 임베디드 S/W를 위한 API 정변환/역변환기의 개발," 대한전기학회 학술대회 논문집, pp.187-189, 2007.
[13] statcounter, https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201810-201910, 2020.11.27.
[14] 조경연, 조한무, 이정국, 서민원, 이상길, 이철훈, "안드로이드에 실시간 성능 제공을 위한 태스크 관리 및 가비지컬렉션 실행 제어 방법," 한국콘텐츠학회논문지, 제18권 제3호, pp.101-113, 2018.
[15] Android developers, https://developer.android.com/guide/platform?hl=ko3, 2020.11.27.
[16] RTDroid, http://rtdroid.cse.buffalo.edu/, 2020.11.27.
[17] A Real-time Extension to the Android Platform, https://embedded.rwth-aachen.de/doku.php?id=en:tools:rtandroid, 2020.11.27.
[18] 이승율, 이상길, 이철훈, "ARM 프로세서 기반의 리눅스를 위한 실시간 확장 커널," 한국콘텐츠학회논문지, 제17권, 제10호, pp.587-597, 2017.
[19] 이상길, 이정국, 이철훈, "리눅스 기반 실시간 성능 제공 RTiK의 이식성 향상을 위한 방법," 한국콘텐츠학회논문지, 제20권, 제8호, pp.54-64, 2020.
[20] RTAndroid Github, https://github.com/RTAndroid, 2021.01.19.
[21] L. Abeni, A. Goel, C. Krasic, J. Snow, and J. Walpole, "A measurement-based analysis of the real-time performance of linux," Real Time and Embedded Technology and Applications Symposium(RTAS), 2002.

저 자 소 개

우 상 호(Sang-Ho Woo) 정회원

- 2018년 2월 : 충남대학교 지질환경과학과(이학사)
- 2020년 2월 : 충남대학교 컴퓨터공학과(공학석사)

〈관심분야〉 : 실시간 운영체제, 임베디드 시스템

이 상 길(Sang-Gil Lee) 정회원

- 2014년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2016년 2월 : 충남대학교 컴퓨터공학과(공학석사)
- 2018년 2월 : 충남대학교 컴퓨터공학과 박사과정 수료

〈관심분야〉 : 실시간 운영체제, 임베디드 시스템

이 철 훈(Cheol-Hoon Lee) 정회원

- 1983년 2월 : 서울대학교 전자공학과(공학사)
- 1988년 2월 : 한국과학기술원 전기 및 전자공학과(공학석사)
- 1992년 2월 : 한국과학기술원 전기 및 전자공학과(공학박사)
- 1983년 3월 ~ 1986년 2월 : 삼성전자 컴퓨터 사업부 연구원
- 1992년 3월 ~ 1994년 2월 : 삼성전자 컴퓨터 사업부 선임연구원
- 1994년 2월 ~ 1995년 2월 : Univ. of Michigan 객원연구원
- 1995년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙연구원

〈관심분야〉 : 실시간 시스템, 운영체제, 고장허용 컴퓨팅, 로봇 미들웨어