

안드로이드에 실시간 성능 제공을 위한 태스크 관리 및 가비지컬렉션 실행 제어 방법

Task Management and Garbage Collection Execution Control Method for Providing Real-time Performance to Android

조경연*, 조한무**, 이정국*, 서민원**, 이상길*, 이철훈*
충남대학교 컴퓨터공학과*, LIG넥스원**

Kyung-Yeon Cho(c-ky10@cnu.ac.kr)*, Han-Moo Jo(hanmoo.jo@lignex1.com)**,
Jeong-Guk Lee(jeongguk.lee@lignex1.com)**, Min-Won Seo(minwon.seo@lignex1.com)**,
Sang-Gil Lee(sk0137@cnu.ac.kr)*, Cheol-Hoon Lee(clee@cnu.ac.kr)*

요약

실시간으로 데이터를 획득하고 평가하는 것이 중요한 군용 점검 장비와 같은 시스템에서는 운영체제 레벨에서 실시간 처리가 가능해야 한다. 기술의 발전으로 기존 장비를 휴대용 단말로 대체하려는 요구가 있으나, 안드로이드가 적용된 휴대용 단말은 실시간성이 요구되는 시스템에 적합하지 않다. 안드로이드에서는 가비지 컬렉션을 통해 가용 메모리를 확보하는데, 이 작업이 수행되는 동안 다른 태스크가 중단되어 특정 태스크의 주기성을 보장할 수 없다. 본 논문에서는 이를 해결하기 위해 안드로이드의 가비지 컬렉션 실행을 제어하는 구조를 설계 및 구현하였다. 실시간 작업이 필요한 시간 동안 가비지 컬렉션을 제어하여 실시간 성능을 보장하며, 안드로이드에 실시간 성능 보장을 위한 RTiK를 적용하였다. 성능 평가를 위해 5ms 주기 태스크의 호출 주기를 측정하였으며, 제어 이전에는 태스크의 34.31%만 주기가 보장되었으나, 제어를 통해 98.18%의 태스크 주기가 만족되어 안드로이드에 실시간성을 제공하였다.

■ 중심어 : | 안드로이드 | 실시간 시스템 | 가비지 컬렉션 |

Abstract

Systems such as military inspection equipment which it is important to acquire and evaluate data in real-time should be able to real-time processing at the operating system level. As technology advances, there is a demand for replacing existing equipment with mobile device, but mobile devices with Android are not suitable for systems requiring real-time performance. On Android, garbage collection ensures free memory, while other tasks are interrupted while this task is performed, which cannot guarantee periodicity of particular tasks. In this paper, we designed and implemented a structure to control execution garbage collection of Android to solve this problem. Real-time performance is ensured by controlling garbage collection during the time required for real-time operation, and RTiK(Real-Time implanted Kernel) is applied to ensure real-time performance on Android. In order to evaluate the performance, we measured the call period of the 5ms period task, and, only 34.31% of the task was guaranteed before the control, but the task period of 98.18% was satisfied through control, providing real-time performance to Android.

■ keyword : | Android | Real-Time System | Garbage Collection |

* 본 연구는 LIG NEX1 산학협력과제 지원으로 연구되었음

접수일자 : 2018년 01월 15일

수정일자 : 2018년 02월 20일

심사완료일 : 2018년 02월 21일

교신저자 : 이철훈, e-mail : cleo@cnu.ac.kr

I. 서론

유도무기체계 사업에서의 점검 장비는 실시간으로 데이터를 획득하고 평가하는 것이 필요하다. 기존의 점검 장비에서는 실시간성 제공을 위해 VxWorks와 같은 상용 실시간 운영체제를 사용하고 있다. 또는 개발의 편의성을 위해 윈도우 운영체제를 사용하는 경우, 실시간 성능 지원을 위해 RTX나 InTime과 같은 별도의 서드파티를 사용한다. 그러나 이러한 제품들은 고가의 구입비용과 라이선스 비용 지불이 필요한 방식이다[1][2]. 최근 하드웨어의 비약적인 발전으로 모바일 단말의 성능이 랩탑을 따라오고 있으며, 스마트폰과 같은 모바일 단말의 활용이 증가함에 따라 군에서는 여러 군용 시스템에서 스마트 모바일 단말을 활용하고자 하는 움직임이 있다[3]. 따라서 모바일 단말을 군용 점검 장비로 활용한다면 기존 점검 장비에 비해 비용을 절감할 수 있다.

모바일 단말에서는 안드로이드 플랫폼이 가장 높은 점유율을 차지하고 있으나[4], 현재 상용 안드로이드는 실시간성이 요구되는 시스템에 적합하지 않다. 그 이유 중 하나는 안드로이드 동작에 기반이 되는 리눅스 커널에서는 모든 쓰레드에 공평한 CPU 점유시간을 제공하기 때문이다. 리눅스 커널은 우선순위 기반 스케줄링과 같은 실시간성을 지원하기 위한 스케줄링 방식을 사용하지 않고 있어, 실시간 태스크가 동작하는데 적합하지 않다.

또한 안드로이드에서는 기기의 성능과 배터리 사용량 간의 균형을 맞추기 위해 CPU 거버너(CPU Frequency Governor)를 통해 CPU 주파수를 동적으로 변경한다. CPU 거버너가 주파수를 변경할 때마다 실행 중인 태스크의 실행 시간에 영향을 미치며, 예측 불가능한 지연 시간이 증가하게 된다.

또한 자바 언어로 작성되는 안드로이드 어플리케이션은 달빅 가상머신(Dalvik Virtual Machine) 상에서 동작하는데, 달빅 가상머신에서는 가용 메모리 확보를 위해 가비지 컬렉션(Garbage Collection, GC)[5]을 이용하여 메모리를 관리한다. 이 때, 가비지 컬렉션 작업을 수행하는 쓰레드 이외에 모든 쓰레드가 중단되는 시간(Stop-the-world)이 존재한다. 가비지 컬렉션으로 인해

중단되는 시간은 정해져 있지 않으며, 그 발생 시점이 메모리 관련 인자 값에 따라 결정되기 때문에 이를 예측하기가 어렵다. 즉, 가비지 컬렉션으로 인해 모든 작업이 중단되어야 하는 시간이 존재하기 때문에, 정해진 주기 내에 실행되어야 하는 실시간 태스크의 동작이 보장되지 않을 수 있다[6][7].

안드로이드에 실시간성을 제공하기 위하여 기존의 플랫폼 구조를 수정한 방법이 있다[8][9]. 이는 기존의 리눅스 커널을 실시간 리눅스 커널로 대체하여, 선점 가능하고 시간 결정적인 우선순위 기반 스케줄링이 가능하게 하였다. 또한 안드로이드 런타임 계층을 실시간 자바 가상머신(Real-Time JVM)으로 대체하거나 실시간 가비지 컬렉션 알고리즘을 적용하여 가비지 컬렉션으로 인한 중단 시간을 해결하였다.

본 논문에서는 안드로이드에 가비지 컬렉션 실행을 제어하는 구조를 설계 및 구현한다. 기존 시스템에서 가비지 컬렉션으로 인한 실시간 태스크의 중단시간을 최소화 할 수 있도록 가비지 컬렉션의 실행을 제어하는 방법을 사용한다. 또한 안드로이드 환경에서 실시간성을 제공할 수 있도록 RTiK[1][10][11]을 개선하여 적용한다. 본 방법을 적용하여 실시간 작업을 필요로 할 경우 가비지 컬렉션 실행을 비활성화 하고, 일반 목적으로 사용될 경우 가비지 컬렉션을 실행할 수 있도록 하는 방법을 제안한다. 이를 통해 안드로이드에 실시간성을 제공할 수 있음을 보인다.

본 논문의 구성은 2장에서 관련 연구를 설명하고, 3장에서 개선된 RTiK과 가비지 컬렉션 실행 제어 구조를 설명한다. 4장에서 실험 방법 및 결과를 제시하고, 5장에서는 결론 및 향후 연구과제에 대해 기술한다.

II. 관련 연구

1. 안드로이드 플랫폼

안드로이드는 모바일 플랫폼으로, 운영체제와 미들웨어 및 핵심 어플리케이션이 포함된 소프트웨어 스택으로 정의된다. 안드로이드 플랫폼은 [그림 1]과 같은 스택 구조를 가진다.

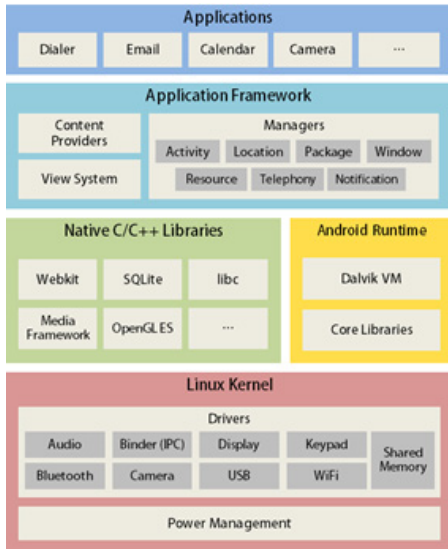


그림 1. 안드로이드 프레임워크 스택

스택 구조의 가장 아래 부분은 리눅스 커널로, 안드로이드 플랫폼의 기반이 된다. 리눅스 커널은 프로세스 및 메모리 관리, 보안 기능과 같은 시스템 기능을 제공한다. 또한 네트워크 인터페이스와 디바이스 드라이버 등을 관리하여 주변 하드웨어와의 인터페이스를 용이하게 하는 역할을 한다.

리눅스 커널 상단에는 C/C++ 라이브러리 집합이 존재하며, 라이브러리에는 안드로이드 환경에 맞게 수정된 C라이브러리(libc)인 Bionic, 오디오 및 비디오, 그래픽, 브라우저, SQLite 등이 포함된다. 특히, Bionic은 CPU 전력이 제한된 모바일 환경에서 빠른 처리를 위해 안드로이드에 최적화된 라이브러이지만, 우선순위 상속을 지원하지 않아 실시간성 제공에 적합하지 않은 단점이 있다.

안드로이드 런타임 시스템은 달빅 가상머신으로, 자바 가상머신을 대신하여 안드로이드에 최적화되어 설계되었다. 달빅 가상머신은 메모리 관리와 멀티 쓰레딩을 위해 리눅스를 완전히 이용한다.

어플리케이션 프레임워크는 자바로 작성된 API 형태로 접근할 수 있으며, 안드로이드 어플리케이션에 다양한 상위 레벨 서비스를 제공한다[12].

2. 달빅 가상머신

자바 언어로 작성된 안드로이드 어플리케이션은 자바 가상머신 상에서 동작해야 한다. 안드로이드에서는 표준 자바 가상머신 대신에 독자적인 달빅 가상머신을 사용한다. 달빅 가상머신은 안드로이드 KitKat(API 19) 버전까지 사용된 런타임 시스템이다. 안드로이드 어플리케이션은 [그림 2]과 같이 달빅 가상머신의 인스턴스 상에서 동작한다. 어플리케이션 마다 별도의 가상머신에서 동작하기 때문에 가비지 컬렉션 또한 독립적으로 발생한다. 달빅 가상머신은 리눅스 커널이 관리하는 프로세스 내에 존재한다[13][14].

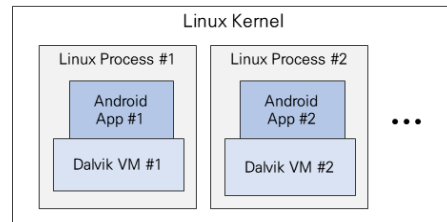


그림 2. 달빅 가상머신과 리눅스 프로세스 간 관계

3. 달빅 가상머신의 가비지 컬렉션

달빅 가상머신에서 수행되는 가비지 컬렉션은 Concurrent Mark-Sweep(CMS) 알고리즘을 사용한다. 이 때 수행되는 가비지 컬렉션의 동작방식은 [그림 3]과 같다. 여기서 뮤테이터 쓰레드란 가비지 컬렉션을 수행하는 쓰레드를 제외한 어플리케이션 쓰레드를 의미한다.

먼저 컬렉션 주기의 시작 부분에서 일괄적으로 모든 뮤테이터 쓰레드들을 중단시킨 후, 힙 루트 스택에서 참조하는 도달 가능한(reachable) 객체들을 찾아서 마킹한다(Initial Mark). 중단시켰던 뮤테이터 쓰레드들을 다시 실행시킨 후에, 이전 단계에서 마킹한 객체가 참조하는 객체들을 따라가면서 도달 가능한지 확인하여 마킹한다(Concurrent Mark). 이 단계가 끝나면 가비지 컬렉터는 다시 모든 뮤테이터 쓰레드들을 중단시킨다. 이전 단계에서의 동시 마킹은 뮤테이터 쓰레드들의 동작과 함께 진행되었기 때문에 새로 추가되거나 참조가 끊긴 객체가 있는지 다시 확인한다(Remark). 그런 다음 뮤테이터 쓰레드들을 다시 시작시키고, 마킹되지 않

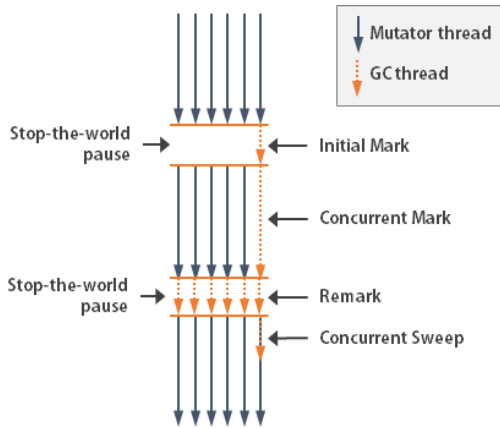


그림 3. Concurrent Mark-Sweep 알고리즘

은 객체들은 가비지로서 정리하는 작업을 수행한다 (Concurrent Sweep)[15]. 이와 같이 가비지 컬렉터에서 CMS 알고리즘을 사용하는 경우, 가비지 컬렉션을 수행하는 쓰레드를 제외한 모든 뮤테이터 쓰레드의 중단이 발생한다. 이 때, 실시간 작업이 필요한 쓰레드가 중단된다면 테드라인 미스(miss)와 같은 문제가 발생할 수 있어 중단 시간을 최소화하기 위한 방법이 필요하다.

표 1. Dalvik VM 가비지 컬렉션 실행 원인

GC Reason	Description
GC_FOR_MALLOC	메모리 할당 요청 시 가용 메모리가 충분하지 않은 경우
GC_CONCURRENT	할당이 CSB 임계 값 초과
GC_EXPLICIT	명시적으로 함수 호출한 경우

달빅 가상머신에서 가비지 컬렉션이 발생하는 원인은 [표 1]과 같다. GC_FOR_MALLOC은 어플리케이션에서 메모리 할당을 요구하였으나, 가용 힙 메모리가 충분하지 않은 경우 발생하는 유형이다. GC_CONCURRENT는 할당된 힙 메모리 크기가 CSB(Concurrent Start Bytes) 임계값을 초과한 경우 발생하며, 뮤테이터 쓰레드가 가비지 컬렉터 데몬에 신호를 보내 백그라운드 GC 사이클을 시작하게 한다. 마지막으로 GC_EXPLICIT은 코드에서 System.gc()를 명시적으로 호출한 경우 발생한다[16].

4. 안드로이드에 실시간성 제공

4.1 RTAndroid

RTAndroid(Real-Time Extension for Android)는 독일 RWTH Aachen University의 Embedded Software Lab에서 개발되었으며, [그림 4]은 RTAndroid의 구조와 기존 안드로이드 구조에서 실시간성 제공을 위해 수정된 부분을 나타내고 있다. 리눅스 커널에는 RT_PREEMPT patch를 적용하여 선점 가능하고 시간 결정적인 우선순위 기반 스케줄링이 가능하도록 한다. 또한 가비지 컬렉션에 Concurrent Reference Counting 알고리즘과 슬랙 기반 스케줄링 방식을 적용하여 연성 실시간성을 제공할 수 있도록 한다[8].

이 방법은 최신 안드로이드 커널 패치가 메인 라인 커널에 아직 통합되지 않았기 때문에, RT_PREEMPT patch를 적용하기 위해 기존 리눅스 커널을 광범위하게 수정해야 하는 단점이 있다[7].

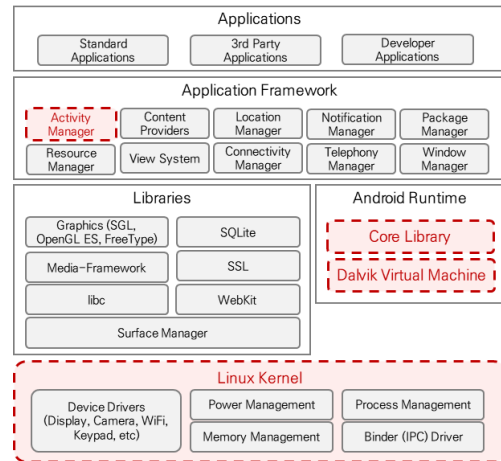


그림 4. RTAndroid 구조

4.2 RTDroid

RTDroid는 안드로이드 애플리케이션에 예측 가능성을 제공하기 위한 목적으로 설계되었다. [그림 5]는 기존 안드로이드 프레임워크 구조에서 수정된 컴포넌트들과 RTDroid 구조를 단순화시켜 나타낸 것이다. 기존의 리눅스 커널을 RT Linux나 RTEMS와 같은 RTOS 커널로 대체하고, 기존의 런타임은 실시간성 제공을 위

하여 실시간 자바 가상머신인 Fiji VM으로 대체한다. 또한, 안드로이드 프레임워크 계층의 Looper, Handler를 비롯한 시스템 서비스들의 구조를 다시 설계하였다. 재설계한 컴포넌트인 RT Looper와 RT Handler에서는 쓰레드가 보내는 메시지에 우선순위를 할당하며, 메시지의 우선순위는 해당 메시지를 보내는 쓰레드의 우선순위를 상속받는다. 이렇게 함으로써 높은 우선순위 쓰레드가 보낸 메시지는 낮은 우선순위 쓰레드가 보낸 메시지에 의해 지연되지 않도록 할 수 있으며, 실시간성을 보장할 수 있다[9].

이 방법의 경우, 안드로이드의 모든 시스템을 처음부터 다시 구축하는 방식으로, 안드로이드 버전의 지속적인 발전에 따른 유지보수가 어려울 수 있다는 단점이 있다[7].

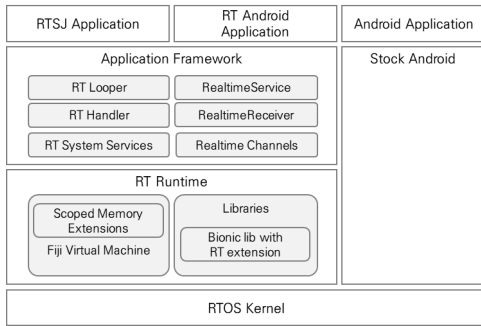


그림 5. RTDroid 구조

III. 안드로이드에 실시간성 제공을 위한 태스크 관리 및 가비지 컬렉션 제어 방법

1. 안드로이드에 실시간성 제어를 위한 방법

안드로이드 운영체제는 리눅스 커널을 사용하며, 안드로이드 프레임워크 실행을 위한 스택 구조를 가진다. 이에 따라서 임베디드 리눅스에 실시간성 제어를 위해 개발한 RTiK-AL를 사용할 수 있으나, 가상 머신에서 동작하는 안드로이드에서 RTiK-AL 모듈 사용을 위한 인터페이스 및 제어 구조를 필요로 하게 된다.

[그림 6]은 안드로이드에 실시간성 제공을 위한 전체 구조를 나타낸 그림이다. 안드로이드의 스택 구조에서

실시간성 제공을 위한 방법으로 RTiK-API's Layer 및 RTiK-APIs Class를 추가하였다.

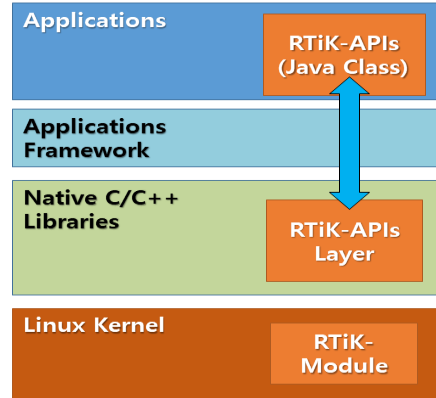


그림 6. 개선된 RTiK의 구조

1.1 RTiK-APIs Layer를 통한 모듈 제어

리눅스 커널에 실시간성 제어를 위해 사용되는 RTiK-Module은 커널 영역에서 실시간 타이머 기능을 제공한다. RTiK-APIs Layer는 이를 안드로이드 어플리케이션에 제공하기 위한 중간 단계의 인터페이스로 제공된다. 리눅스 커널 모듈로 제공되는 RTiK-Module 제어를 위해 ioctl을 통한 커널에 접근할 수 있도록 다음의 함수를 제공한다. [표 2]는 모듈 제어 함수를 위해 제공하는 RTiK-APIs Layer의 함수 목록을 나타낸 표이다.

표 2. RTiK-APIs Layer의 모듈 제어 함수

기능	함수명	설명
태스크 제어	rtik_core_AddProc()	실시간 태스크 관리를 위해 모듈에 등록하는 함수
	rtik_core_DelProc()	사용이 끝난 태스크의 등록 해제를 위한 함수
	rtik_core_recvAck()	태스크 응답을 위한 함수
타이머 제어	rtik_core_StartTimer()	실시간 타이머 시작 명령 함수
	rtik_core_EndTimer()	실시간 타이머 중지 명령 함수

RTiK-Module을 통해 태스크의 실시간 성능 제공을 위해 태스크 제어를 위한 모듈함수 3개를 제공하고 있다. 실시간 태스크 등록 및 사용 해제 함수와 시스템의

오버헤드를 줄이기 위한 태스크 응답 확인 함수를 제공하고 있다.

또한, 태스크 등록 후에 실시간 타이머를 동작시켜 RTiK-Module의 시작 및 종료를 명령할 수 있도록 하는 2개의 함수를 제공한다. 이를 통해서 사용자 영역에 실시간성을 제공할 수 있도록 하는 인터페이스 함수가 제공된다. 안드로이드 운영체제에서는 사용자 어플리케이션에 cpp 언어로 작성된 인터페이스 함수를 사용하도록 NDK(Native Development Kit)을 통해 Native C/C++ Library 영역에 RTiK-APIs Layer를 등록하여 안드로이드 사용자 영역에서 접근할 수 있도록 한다.

1.2 RTiK-APIs Class를 통한 안드로이드 어플리케이션에서의 제어

안드로이드 운영체제는 가상 머신에서 동작하는 어플리케이션을 통해 사용자에게 서비스를 제공한다. 이에 따라 실시간 모듈을 제어하기 위해서는 자바로 작성된 클래스를 필요로 하게 된다. 이 자바 클래스는 1.1.절에서 작성한 RTiK-APIs Layer의 Native Library를 통해 모듈에 접근할 수 있는 함수로 제공된다.

[그림 7]은 cpp 언어로 작성된 RTiK-APIs Layer 접근을 위하여 자바와 cpp 간의 정보 전달을 위한 JNI(Java Native Interface) 함수를 나타낸다. System.loadLibrary() 함수를 통해 라이브러리를 메모리에 읽어주며, 접근을 위한 4개의 함수는 [표 3]과 같다.

```
// Used to load the 'native-lib' library on application startup.
static {
    System.loadLibrary( libname: "rtik-native-lib");
}

/**
 * A native method that is implemented by the 'native-lib' native library,
 * which is packaged with this application.
 */

public native int _startTimer();
public native int _stopTimer();
public native int _initialize();
public native int _joinResult();
```

그림 7. Native Library 접근을 위한 JNI인터페이스

표 3. JNI 함수 목록

기능	함수명	설명
초기화	_initialize()	JNI 인터페이스 초기화
타이머 관리	_startTimer() _stopTimer()	타이머 시작 명령 함수 타이머 종료 명령 함수
태스크 관리	_joinResult()	실시간 태스크 종료응답 함수

초기화 함수를 이용하여 JNI에서 제공하는 Native Library의 함수 내부에 있는 기본 자료구조를 초기화한다. 또한, 타이머 관리를 위한 명령 함수를 제공하며, 실시간 태스크의 실행 동기화를 위한 응답 대기 함수를 제공하고 있다.

[그림 8]은 JNI를 통해 cpp 라이브러리인 RTiK-APIs Layer의 함수를 호출하는 예시를 나타낸다. 이를 통해 자바 영역과 cpp 영역을 연결하여 안드로이드 어플리케이션에서도 모듈을 제어할 수 있도록 할 수 있게 된다.

```
extern "C"
JNIEXPORT int JNICALL
Java_kr_ac_cnu_sslab_testjniproject_RTik_LAPI_1startTimer(
    JNIEnv *env,
    jobject /* this */) {

    int result = _rtik_startTimer();

    return result;
}
```

그림 8. JNI를 통해 cpp 함수를 호출하는 방법

1.3 실시간 태스크 제공 방법

기존 안드로이드의 자바 스레드는 우선순위를 제공하고 있으나, 이는 리눅스에서 Nice 값을 이용한 동적 우선순위 방법을 이용하는 것으로 실시간 태스크에 사용할 수 없는 한계를 보인다. 이를 해결하기 위해 Pthread 라이브러리를 이용한 Native Library 영역에 태스크를 생성한다.

```
int* rtik_task_create ( rtHandle handle, int nResol )
{
    struct rtTaskInfo rtTaskInfo;
    rtTaskInfo.handler = handle;
    rtTaskInfo.nResol = nResol;

    pthread_attr_t attr;
    if ( pthread_attr_init ( &attr ) != 0 ) return 1;
    if ( pthread_create ( &tid, &attr, rtBasicHandle, (void *)&rtTaskInfo ) != 0 )
        return -1;

    return 0;
}
```

그림 9. 실시간 태스크 생성을 위한 함수

rtik_task_create() 함수를 통해서 pthread를 통한 스레드를 생성한다. 이 때, 우선순위를 조정하여 실시간 태스크를 위한 설정을 수행한다. 또한, 다양한 사용자의 함수에 대한 호출을 위해 함수를 등록할 수 있도록 인자를 통해 제공한다.

```
typedef int (*rtHandle) (int, int);
struct _rtTaskInfo {
    rtHandle      handler;
    int           nResol;
    int           arg1;
    int           arg2;
};

int rtk_task_create( rtHandle handle, int nResol );
```

그림 10. 사용자 정의 함수 수행을 위한 함수 포인터 제공

[그림 10]은 사용자 함수 등록을 위한 자료구조를 나타낸다. 매개변수가 2개인 함수 타입을 임의로 사용자가 지정할 수 있도록 하여, 사용자가 작성하는 다양한 함수가 실행될 수 있도록 한다. 이를 통해서 사용자 영역에 실시간 태스크를 제공할 수 있도록 한다.

2. 달빅 가상머신의 가비지 컬렉션 실행 제어 방법

2.1 가비지 컬렉션 실행 제어 구조

가비지 컬렉션으로 인한 중단 시간을 최소화 하고자 실행 제어를 통해 가비지 컬렉션이 실행되지 않으면, 힙 메모리에 가비지가 남아있게 되고 해당 메모리 공간은 다른 객체에 의해 사용될 수 없다. 이에 따라 메모리 할당이 요청될 때마다 가용 메모리가 부족한 현상이 빈번하게 발생하며, 힙 메모리의 크기가 지속적으로 증가한다. 그렇기 때문에 가비지 컬렉션 실행이 비활성화되는 시간은 실시간 태스크가 실행되는 특정 시간으로 한정시켜야 한다. 그 이외의 상황에서는 가비지 컬렉션이 실행될 수 있도록 하는 것이 필요하다.

본 논문에서 제안하는 가비지 컬렉션 실행 제어 구조는 안드로이드의 달빅 가상머신을 수정한 것으로, 달빅 가상머신에서 가용 메모리 확보를 위해 가비지 컬렉션을 실행해야 하는 시점에서 특정 플래그 값에 따라 그 실행 여부가 결정된다.

안드로이드 어플리케이션은 리눅스 프로세스에 대응되며, 각자 자신만의 달빅 가상머신 인스턴스 상에서 실행된다. 달빅 가상머신은 Zygote 프로세스에 의해 생성 및 초기화되어 구동된다. Zygote 프로세스는 어플리케이션의 실행에 필요한 코드 및 메모리 정보들을 미리 메모리에 적재 해두었다가 새로운 어플리케이션이 실행될 때 빠르게 실행시키는 역할을 한다.

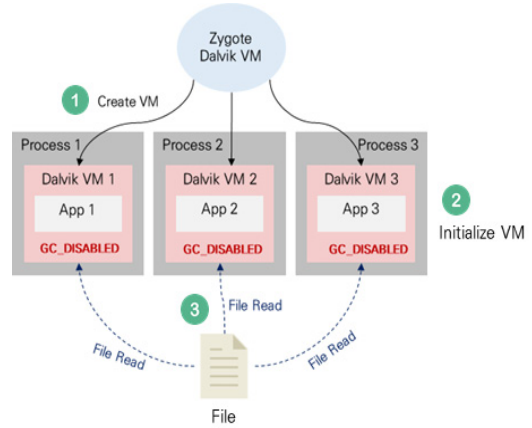


그림 11. 달빅 가상머신 초기화 시 가비지 컬렉션 여부 설정 과정

[그림 11]은 달빅 가상머신이 초기화될 때 가비지 컬렉션 여부를 설정하는 과정을 나타낸 것이다. Zygote 달빅 가상머신에 의해 시스템에 존재하는 모든 프로세스의 달빅 가상머신 인스턴스가 생성되고 초기화되며, 이 과정에서 가비지 컬렉션의 실행 여부가 결정되도록 하였다. 이를 위해 가비지 컬렉션 실행 여부를 저장하고 있는 파일을 하나 두었으며, 달빅 가상머신 코드 상에서 파일에 접근할 수 있도록 하기 위해 리눅스 파일 시스템에 위치시켰다. 달빅 가상머신이 초기화 되는 과정에서 해당 파일에서 값을 읽어와 GC_DISABLED라는 플래그 변수에 설정한다. 이와 같은 과정을 통해 안드로이드 시스템 내에 존재하는 모든 달빅 가상머신의 가비지 컬렉션 실행이 제어된다.

2.2 가비지 컬렉션 실행 제어 동작 과정

[그림 12]는 달빅 가상머신이 초기화될 때 가비지 컬렉션 실행 여부에 대한 플래그 값이 설정된 후, 가비지 컬렉션 실행 여부가 결정되는 과정을 나타낸다. 가비지 컬렉션이 필요한 시점에서 GC_DISABLED 변수 값에 따라 실행 여부가 결정된다. GC_DISABLED 변수는 달빅 가상머신의 가비지 컬렉션 실행 여부를 나타내는 변수로, 값이 1인 경우 가비지 컬렉션을 실행하지 않고, 0인 경우 실행한다는 의미를 가진다.

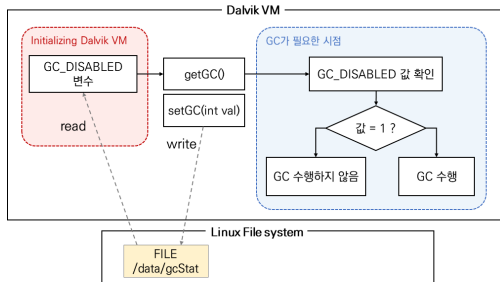


그림 12. 달빅 가상머신의 가비지 컬렉션 실행 제어 과정

달빅 가상머신은 가비지 컬렉션이 필요한 시점에서 `dvmCollectGarbageInternal` 함수를 호출하여 도달할 수 없는 객체를 가비지로 식별하고, 가비지가 차지하고 있는 메모리 공간을 회수한다.

기존의 달빅 가상머신 코드에 가비지 컬렉션의 실행을 제어하기 위한 함수를 구현하였으며, 구현한 함수는 [표 4]와 같다.

표 4. 가비지 컬렉션 실행 제어 함수

함수명	설명
<code>getGCstatus</code>	GC_DISABLED 변수의 getter 함수
<code>setGCstatus</code>	GC_DISABLED 변수의 setter 함수
<code>dvmGcExecSetup</code>	파일에 값을 기록하는 함수
<code>dvmGcExecGetValue</code>	파일에서 값을 읽어 GC_DISABLED 변수 설정하는 함수

1) `getGCstatus`, `setGCstatus` 함수

`getGCstatus()` 함수는 GC_DISABLED 변수에 대한 getter 함수로, 변수 값을 리턴하며, `setGCstatus()` 함수는 GC_DISABLED 변수의 setter 함수이다. 해당 변수 값은 파일에서 값을 읽어서 설정되기 때문에, 인자로 전달 받은 값을 파일에 기록하기 위해 `dvmGcExecSetup` 함수를 호출하는 작업을 수행한다.

2) `dvmGcExecSetup` 함수

가비지 컬렉션 실행 여부를 파일에 설정하는 함수로, 인자로 전달 받은 값을 파일에 기록한다.

3) `dvmExecGetValue` 함수

파일에서 값을 읽어서 GC_DISABLED 변수에 해당 값을 설정하는 함수이다.

IV. 실험 환경 및 결과

1. 실험 환경 및 실험 방법

본 논문은 달빅 가상머신의 가비지 컬렉션의 실행 여부에 따라 실시간성 제공에 미치는 영향을 분석하기 위해 UDP를 통한 소켓 통신이 이루어지는 상황을 가정하였다. [그림 13]와 [표 5]는 실험 환경과 동작 시나리오를 나타낸 것이다. 실험을 위하여 Nexus 5 기기에 달빅 가상머신 코드가 수정된 안드로이드 AOSP 4.4.2 버전을 탑재하였다.

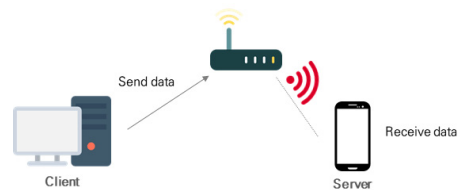


그림 13. 실험 환경 및 동작 시나리오

표 5. 실험 환경

분류	Server	Client
Device	Nexus 5	PC
OS	AOSP Android 4.4.2 Linux Kernel 3.4.0	Windows 10
CPU	Qualcomm MSM 8974 Hammerhead ARMv7 Processor	Intel Core i7-2600 CPU @3.40 GHz
Memory	2GB	
Compile	Dalvik VM	Visual Studio 2010

소켓 통신에서 클라이언트 프로그램은 C언어로 작성되었으며, 10ms 주기로 1KB 크기의 데이터를 서버에 보내는 동작을 수행한다. 안드로이드 어플리케이션은 통신 서버로 동작하며, 공유기를 통해 외부에서 들어오는 데이터를 수신하여 ArrayList와 배열과 같은 특정 자료구조에 저장한다. 소켓 통신은 최대 30분 동안 이루어진다고 가정하였으며, 실험은 다음과 같이 진행하였다. 첫 번째 실험은 가비지 컬렉션 제어에 따른 실시간 성능 차이를 측정한다. 두 번째 실험은 가비지 컬렉션이 실행되는 환경과 실행되지 않는 환경에서 시간에

따라 변화하는 힙 메모리 크기를 측정한다. 힙의 증가에 따라 메모리 문제가 발생하지 않음을 확인한다. 세 번째 실험으로는 소켓 통신을 통해 수신한 데이터를 안드로이드 어플리케이션에서 특정 자료구조에 추가하는데 소요되는 시간을 측정한다.

2. 실험 결과

2.1 가비지 컬렉션 제어에 따른 실시간 성능 차이

본 논문에서 제안한 가비지 컬렉션 실행 제어에 따른 실시간 성능 차이를 확인하기 위해 제어 방법이 적용되지 않은 환경과 가비지 컬렉션이 적용된 상황에서의 실험을 통해 비교한다. 실험은 5ms 주기를 가지는 태스크의 주기성 만족을 확인하기 위해 가비지 컬렉션 실행 제어를 통한 비활성화와 활성화 시 태스크 호출 주기의 값을 비교한다.

[그림 14]와 [표 6]은 가비지 컬렉션 제어를 하지 않아 가비지 컬렉터가 활성화 되어 있을 때의 태스크의 호출 주기를 측정한 실험 결과이다.

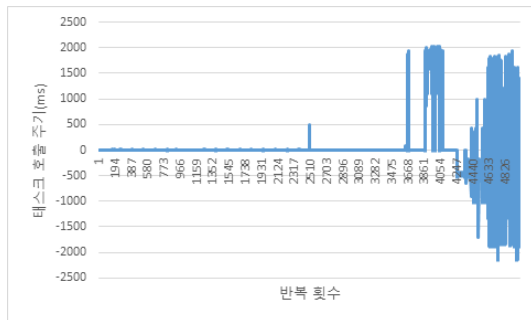


그림 14. 가비지 컬렉션 제어 전 호출 주기 그래프

표 6. 가비지 컬렉션 제어 전 측정 결과

태스크 호출 주기	측정횟수	
	횟수	횟수/총횟수 (%)
4.5 ~ 5.5 ms	1715 회	34.31 %
< 4.5 ms	2120 회	42.41 %
> 5.5 ms	391 회	7.82 %
> 6.0 ms	387 회	7.74 %
> 10.0ms	386 회	7.72 %

가비지 컬렉션을 제어하기 이전의 데이터는 10ms 이상을 벗어나는 횟수가 386회로 7.72 %를 보이고 있으며, 4.5 ms 이하로 벗어나는 호출 주기는 2120 회, 42.41%로 나타나고 있다. 반면 10% 오차 범위 이내인 4.5 ms ~ 5.5ms 주기 범위는 1715 회인 34.31 %를 나타낸다. 이런 실험 결과를 통해 가비지 컬렉션을 활성화 하게 되면 실시간성을 전혀 만족할 수 없게 된다는 것을 보이고 있다.

[그림 15]와 [표 7]은 가비지 컬렉션 제어를 통해 실시간성이 요구되는 시간동안 가비지 컬렉터가 비활성화 되어 있을 때의 태스크 호출 주기를 측정한 실험 결과이다.

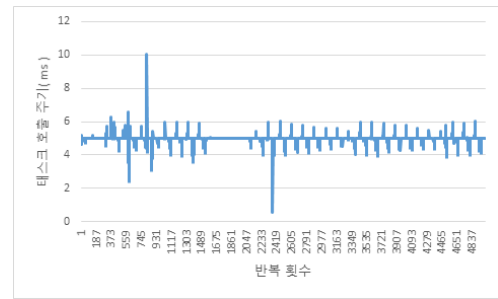


그림 15. 가비지 컬렉션 제어 후 호출 주기 그래프

표 7. 가비지 컬렉션 제어 후 측정 결과

태스크 호출 주기	측정횟수	
	횟수	횟수/총횟수 (%)
4.5 ~ 5.5 ms	4908 회	98.18 %
< 4.5 ms	54 회	1.06 %
> 5.5 ms	32 회	0.64 %
> 6.0 ms	5 회	0.10 %
> 10.0ms	1 회	0.02 %

가비지 컬렉션을 비활성화 하여 기존에 발생하던 10 ms를 초과하는 태스크 호출 주기는 1회로 감소하였으며, 그 외에 호출 주기를 보이는 데이터 또한 데이터의 오차범위 10% 이내인 4.5 ms ~ 5.5 ms 주기 범위로 나타났을 때, 유효 횟수가 4908회인 98.18 %로 나타났다. 이를 통해서 기존 가비지 컬렉터를 활성화 할 때보다 크게 감소하는 것으로 이런 가비지 컬렉터를 제어하

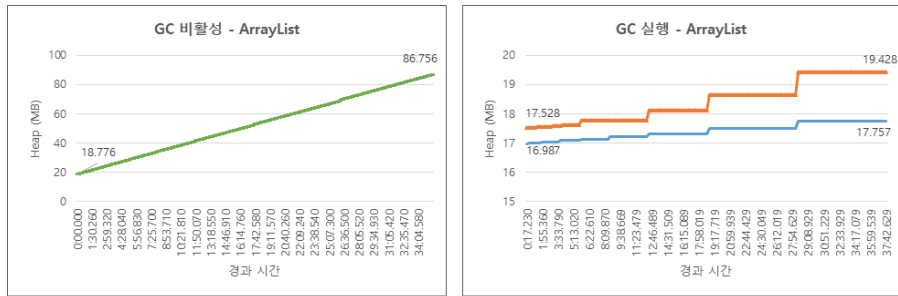


그림 16. 시간에 따른 힙 크기 변화 - ArrayList

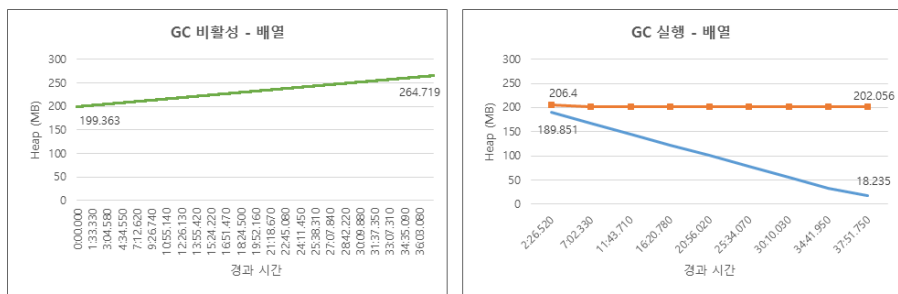


그림 17. 시간에 따른 힙 크기 변화 - 배열

는 것이 실시간성을 만족하도록 할 수 있는 것을 나타낸다.

2.2 시간에 따른 힙 메모리 크기

두 번째 실험으로 10ms 주기의 UDP 소켓 통신이 30분간 이루어지는 동안 시간에 따른 힙 메모리 크기를 측정한다. 가비지 컬렉션 실행 여부 및 사용한 자료구조에 따라 힙 메모리 크기는 다음 그림과 같다. 측정된 힙 크기를 [표 8]과 같이 요약할 수 있다. [그림 16]은 ArrayList를 사용한 경우 측정된 데이터를 그래프로 나타낸 것이고, [그림 17]은 배열을 사용한 경우의 그래프이다. 측정된 데이터 x축은 힙 메모리가 증가한 시점을 나타내며, y축은 MB 단위의 힙 메모리 크기이다.

데이터 수신을 위해 ArrayList와 배열을 사용한 경우 모두 실행 제어를 통해 가비지 컬렉션이 발생하지 않는 상황에서는 가용 메모리가 충분히 확보되지 않기 때문에, 지속적으로 힙의 크기가 증가한다. 그러나 30분간의 통신이 이루어지는 동안 가비지 컬렉션이 실행되지 않는 것으로 인한 메모리 관련 오류는 발생하지 않는 것

을 확인하였다. 가비지 컬렉션이 실행되는 상황에서는 할당에 필요한 가용 메모리 확보를 위해 가비지 컬렉션이 일어나게 되고, 이에 따라 힙 메모리가 비교적 크게 증가하지 않는다.

표 8. 가비지 컬렉션 실행 여부 및 자료구조에 따른 힙 크기 (MB)

	ArrayList		배열	
	GC disabled	GC enabled	GC disabled	GC enabled
초기	18,776	17,528	199,363	206,4
통신 후	85,857	19,428	265,275	202,056

2.2 데이터 추가 소요 시간 측정

세 번째 실험은 가비지 컬렉션 실행 여부 및 수신한 데이터 저장에 사용한 자료구조에 따른 데이터 추가 시간을 비교하였으며, 실험 결과는 [표 9]와 같이 요약할 수 있다.

표 9. 가비지 컬렉션 실행 여부 및 자료구조에 따른 수신 데이터 추가 시간

	ArrayList		배열	
	GC disabled	GC enabled	GC disabled	GC enabled
최대값(ms)	1.69	69.64	1.92	8.22
최소값(μ s)	1.66	1.56	1.4	1.45
평균값(μ s)	4.29	13.24	3.42	10.7

ArrayList를 사용했을 때 가비지 컬렉션이 실행되는 경우보다 실행되지 않는 경우가 평균적으로 약 3.08배 빠르며, 최악의 소요 시간은 가비지 컬렉션이 실행되지 않는 경우가 약 4.12배 빠르다. 또한 수신 데이터를 배열에 추가하는 경우, 가비지 컬렉션이 실행되지 않는 경우가 평균적으로 약 3.13배 빠르며, 최악의 시간은 약 4.28배 빠른 것을 확인하였다. 이와 같이 실행 제어를 통해 가비지 컬렉션이 실행되지 않는 경우, ArrayList와 배열 자료구조에 데이터를 추가하는데 걸리는 평균 시간 및 최대 시간이 감소하게 되는 것을 확인하였다.

V. 결론 및 향후 연구과제

안드로이드 어플리케이션은 달빅 가상머신 상에서 실행되며, 가비지 컬렉션을 통해 자동 메모리 관리가 수행된다. 그러나 가비지 컬렉션으로 인한 중단시간은 안드로이드의 실시간성 제공에 있어 시간 결정성을 보장할 수 없는 원인으로 지목된다. 이를 해결하여 실시간성이 제공되는 안드로이드에서 주기 성능을 만족하기 위해 본 논문에서는 안드로이드 달빅 가상머신의 가비지 컬렉션 실행 자체를 제어하는 구조를 설계 및 구현하였다. 또한 자바 API를 구현하여 사용자가 어플리케이션을 통해 필요에 따라 가비지 컬렉션의 실행을 활성 또는 비활성 할 수 있는 기능을 제공하였다.

가비지 컬렉션을 제어하기 이전에는 5ms 주기의 태스크의 오차범위 10%인 4.5ms ~ 5.5ms 주기 범위에서 총 4999회 중 1715회 인 34.31%만을 오차범위 이내에서 호출되는 것으로 실시간성을 만족할 수 없었으나, 본 논문에서 제안한 방법을 통해 가비지 컬렉터를 제어한

이후 오차범위 내에서 4908회 인 98.18%의 태스크가 호출 주기를 만족하는 것을 보였다.

또한 가비지 컬렉션이 비활성화 되어 있을 때, 메모리에 접근하여 데이터를 추가하는 시간을 확인하였으며, ArrayList를 사용할 경우 비활성화 시 최소 1.66 ns, 최대 1.69 μ s, 평균 4.29 μ s를 나타내어, 시간이 매우 적게 소요되는 것으로 확인되었다. 이를 통해 안드로이드에 실시간성을 제공하기 위해서 가비지 컬렉션 제어를 사용하여 실시간 태스크의 주기를 만족할 수 있었으며, 또한 이때 메모리 접근 시간을 줄일 수 있어서 실시간 태스크 성능에 거의 영향이 없도록 할 수 있었다.

향후 연구로는 안드로이드에서 실시간 태스크를 통해 다양한 기능을 구현할 수 있도록 여러 가지 서비스에 대한 지원기능을 연구해야 하며, 안드로이드에서 실시간 태스크의 호출 주기를 낮추기 위한 방법에 대한 연구가 요구된다.

참 고 문 헌

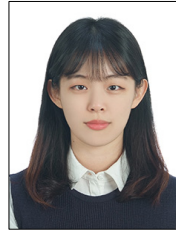
- [1] 이진욱, 조문행, 김종진, 조한무, 박영수, 이철훈 “윈도우 기반의 점점장비에 실시간성을 지원하는 실시간 이식 커널의 설계 및 구현,” 한국콘텐츠학회논문지, Vol.10, No.10, pp.36-44, 2010.
- [2] 송창인, 이승훈, 주민규, 이철훈, “멀티프로세서 윈도우즈 상에서 실시간성 지원,” 한국콘텐츠학회논문지, Vol.12, No.6, pp.68-77, 2012.
- [3] 손익재, 김일호, 양종휴, 이남용, “사이버 국방을 위한 스마트 단말 보안기술,” 한국통신학회논문지, 제13권, 제10호, pp.986-992, 2012.
- [4] <https://www.netmarketshare.com/>
- [5] R. Jones, A. Hosking, and M. Eliot, *The Garbage Collection Handbook: the art of automatic memory management*, CRC Press, 2016.
- [6] 김도현, 강형석, 강정남, 이은규, 김강희, “실시간 정밀 모션 제어를 위한 안드로이드 응용 설계 및 구현,” 정보과학회 컴퓨팅의 실제 논문지, 제21권, 제4호, pp.315-319, 2015.

- [7] A. P. Ruiz, M. A. Rivas, and M. G. Harbour, "CPU Isolation on the Android OS for running Real-Time Applications," Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems, ACM, 2015.
- [8] T. Gerlitz, I. Kalkov, J. Schommer, D. Franke, and S. Kowalewski, "Non-blocking garbage collection for real-time android," In Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems, pp.108-117, 2013.
- [9] Y. Yan, S. Cosgrove, V. Anand, A. Kulkarni, S. Konduri, S. Ko, and L. Ziarek, "RTDriod: A Design for Real-Time Android," IEEE Transactions on Mobile Computing, Vol.15, No.10, pp.2564-2584, 2016.
- [10] 이상길, 이승율, 이철훈, "리눅스 사용자 영역에 실시간성 제공을 위한 미들웨어," 한국콘텐츠학회논문지, Vol.16, No.5, pp.217-228, 2016.
- [11] 이승율, 이상길, 이철훈, "ARM 프로세서 기반의 리눅스를 위한 실시간 확장 커널," 한국콘텐츠학회논문지, Vol.17, No.10, pp.587-597, 2017.
- [12] <https://developer.android.com/guide/platform/index.html>
- [13] <https://www.ibm.com/developerworks/opensource/library/os-android-devel/>
- [14] <http://higes.com/android-security-part-1/>
- [15] Sun Microsystems, *Memory Management in the Java HotSpot™ Virtual Machine*, Sun Microsystems, 2006.
- [16] A Hussein, M Payer, and A Hosking, "Impact of GC design on power and performance for Android," Proceedings of the 8th ACM International Systems and Storage Conference, ACM, 2015.

저 자 소 개

조 경 연(Kyung-Yeon Cho)

정회원



- 2016년 2월 : 충남대학교 컴퓨터 공학과(공학사)
- 2018년 2월 : 충남대학교 컴퓨터 공학과(공학석사)

<관심분야> : 운영체제, 실시간 시스템, 임베디드 시스템, 안드로이드 플랫폼

조 한 무(Han-Moo Jo)

정회원



- 1995년 2월 : 숭실대학교 전자공학과(공학사)
- 1994년 ~ 현재 : LIG넥스원 수석연구원

<관심분야> : 유도무기, 점검장비, 임베디드 시스템

이 정 국(Jeong-Guk Lee)

정회원



- 2011년 2월 : 인하대학교 정보통신공학과(공학사)
- 2012년 ~ 현재 : LIG넥스원 선임연구원

<관심분야> : 유도무기, 점검장비, 임베디드 시스템

서 민 원(Min-Won Seo)

정회원



- 2013년 2월 : 한국항공대학교 항공우주공학과(공학사)
- 2015년 2월 : 한국과학기술원 항공우주공학과(공학석사)
- 2015년 ~ 현재 : LIG넥스원 선임연구원

<관심분야> : 유도조종 알고리즘, 점검장비, 임베디드 시스템

이 상 길(Sang-Gil Lee)

정회원



- 2014년 2월 : 충남대학교 컴퓨터 공학과(공학사)
- 2016년 2월 : 충남대학교 컴퓨터 공학과(공학석사)
- 2016년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 박사과정 재학

<관심분야> : 실시간 운영체제, 임베디드 시스템

이 철 훈(Cheol-Hoon Lee)

정회원



- 1983년 2월 : 서울대학교 전자공학(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 객원 연구원
- 1995년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙 연구원

<관심분야> : 실시간 시스템, 운영체제, 고장허용 컴퓨팅, 로봇 미들웨어