다수의 어플리케이션을 위한 멀티 스레드 센서 프레임워크

황인중○ 신동군
성균관대학교 전자전기컴퓨터공학과
sinban@skku.edu, dongkun@skku.edu

Multi-threaded Sensor Framework for Multiple Applications

Injung Hwang○ Dongkun Shin
Department of ECE, Sungkyunkwan University

요 약

최근에 헬스 케어, 스마트 홈, 스마트 시티, 빅 데이터, 스마트 파밍 같은 사물인터넷 서비스들이 큰 화제를 불러일으키며, 많은 기업들이 사물인터넷 서비스 개발에 매진하고 있다. 위와 같은 여러 사물인터넷 서비스들의 예를 볼 때, 사물인터넷에서 주요한 기능은 주어진 환경에서 센서 기기들을 통해 주변의 센서 데이터를 수집하고, 수집된 데이터를 기반으로 가치 있는 정보를 추출하는 일이라고 할 수 있다. 사물인터넷은 모든 사물들이 인터넷으로 연결되어 모바일 환경에 비해 센서 데이터 수집량도 훨씬 증가했으며 그 활용도도 증가했다. 즉, 센서 기기들을 소수의 어플리케이션이 사용했던 모바일 환경과는 달리, 사물인터넷 환경에서는 훨씬 더 많은 어플리케이션들이 센서 기기들을 접근한다. 모바일 환경에서는 소수의 어플리케이션이 센서 데이터에 대해 접근했기 때문에 센서 요구 이벤트에 따라 스레드를 생성하여 센서 데이터를 전송하는 방식을 사용했으나, 이 방법을 사물인터넷 환경에서 사용할 경우 다수의 어플리케이션이 센서 요구 이벤트를 발생시켜 비효율적으로 너무 많은 스레드가 생성될 가능성이 있다. 이에 따라 본 논문에서는 다수의 어플리케이션을 위해 센서 마다 스레드를 할당하는 방식의 멀티 스레드 센서 프레임워크를 제안한다.

1. 서 론

최근 모바일 기기가 급격하게 발전함에 따라 소형 플랫폼 기기들도 함께 발전하였고, 이로 인해 모든 기기들이 인터넷에 연결되는 사물인터넷에 대한 관심이 크게 증가하고 있다. 네스트 랩스의 네스트, 삼성의 스마트 싱스 같은 스마트홈, 나이키, 저우본 등 헬스 케어, 빅데이터 등 다양한 분야에서 사물인터넷이 활용되고 있다. 이러한 서비스들의 주요 역할은 집안, 사람의 신체 등 주변 환경에서 센서 데이터를 수집하여, 해당 센서 데이터를 활용하여 사람들에게 서비스를 제공하는 것이다. 사물인터넷에서는 의자, 책상, 문 등 모든 사물들이 센서로 활용될 수 있고, 모든 기기들이 인터넷에 연결되어 여러 어플리케이션이 여러 센서 데이터를 공유한다. 그러므로 다수의 어플리케이션이 센서 데이터에 접근하는 일이 발생할 수 있다.

기존 모바일 환경에서는 모바일의 센서 관리자가 어플리케이션에게서 센서에 대한 요구 이벤트를 받으면 요구 이벤트마다 스레드를 생성하여 센서 데이터를 수집, 전달하였다. 하지만 이러한 방식을 다수의 어플리케이션이 작동하는 사물인터넷에서 사용할 경우, 여러 어플리케이션들의 센서 데이터 요구 이벤트가 증가함에 따라서, 각 요구 이벤트에 대한 스레드 수가 증가한다. 이는 매우 비효율적이며, 이벤트 수가 증가함에 따라서 성능 저하가 발생할 수 있다. 따라서, 본 논문에서는 사물인터넷 환경에서 여러 다수의 어플리케이션들에게 센서 데이터를 효율적으로 전달하기 위해 각 센서 마다 스레드를 할당하는 멀티 스레드 기반 센서 프레임워크를 제안한다.

또한 사물인터넷 환경의 서비스 개발자들을 위해 자바스크립트 언어로 어플리케이션 개발이 가능하도록 개발 환경을 제공하였다. 자바스크립트는 다른 언어들에 비해 배우기 쉬워, 현재 가장 큰 개발자 커뮤니티인 Stackoverflow 와 오픈소스 코드관리 사이트인 Github에서 가장 많이 사용되는 언어이므로 더 많은 서비스 개발자들을 지원할 수 있다. [1]

2. 관련 연구

2.1 안드로이드 센서 프레임워크

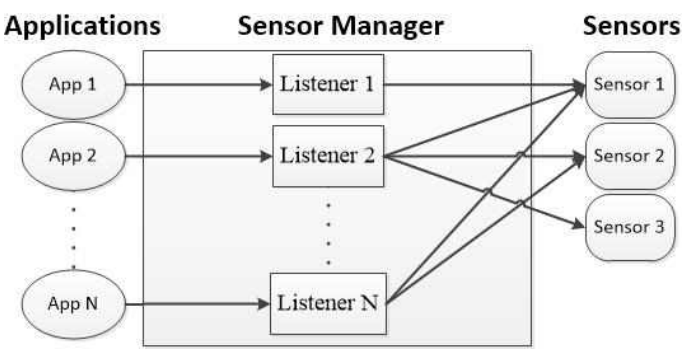

그림 1 안드로이드 센서 관리자의 센서 수집 방식

이 논문은 2013년 정부(미래창조과학부)의 재원으로 (재)스마트IT융합시스템 연구단(글로벌프론티어사업)의 지원을 받아 수행된 연구임. ((재)스마트IT융합시스템 연구단-2011-0031863)

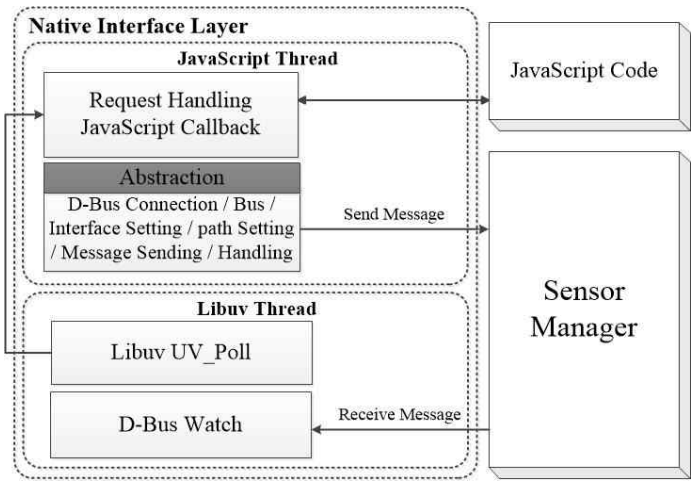

그림 2 제안하는 프레임워크의 전체 구조

현재 모바일 환경에서 가장 많이 사용되고 있는 운영체제는 안드로이드이다. 그림 1는 안드로이드에서 사용되고 있는 센서 프레임워크에 대한 대략적인 그림이다. 안드로이드 어플리케이션은 센서 데이터를 수집하기 위해서 리스너라는 오브젝트를 사용한다. 각 어플리케이션은 원하는 센서 데이터에 대한 정보와 함께 리스너를 센서 관리자에 등록하면, 센서 관리자에서는 해당 센서에 대한 디바이스 드라이버를 통해 각 센서 데이터를 수집하여 이벤트 작동 방식 혹은 주기적인 방식으로 센서 데이터를 어플리케이션에게 제공한다. [2]

예를 들어, 그림 1에서는 App 1이 센서 관리자에 리스너 1, 2를 등록하고, 각 리스너가 sensor 1, sensor 2, 3의 데이터를 수집하는 과정을 보여준다. 이와 같은 방식은 어플리케이션이 많아짐에 따라서 리스너 수가 많아지게 되고, 이는 같은 센서에 대해 여러 리스너가 데이터 수집을 동시에 요청하는 문제를 초래할 수 있다.

2.2 타이젠 센서 프레임워크

타이젠은 C언어로 된 네이티브 어플리케이션과 자바스크립트로 작성된 어플리케이션을 모두 실행시킬 수 있는 구조를 가지고 있다. 네이티브 어플리케이션과 자바스크립트로 작성된 웹어플리케이션 모두 센서 데이터를 수집할 수 있다.

각 어플리케이션들은 소켓을 통해 센서 관리자와 통신하며, 센서 관리자는 해당 요청들을 받아들여서 워커 스레드를 통해 실제 센서의 데이터를 수집한다. [3] 즉, 요구 이벤트가 발생함에 따라서 요구 이벤트마다 워커 스레드가 생성된다. 그러므로 타이젠 또한 여러 어플리케이션에서 요구 이벤트를 더 많이 발생함에 따라서 워커 스레드의 수가 계속해서 증가하게 되어 비효율적이다.

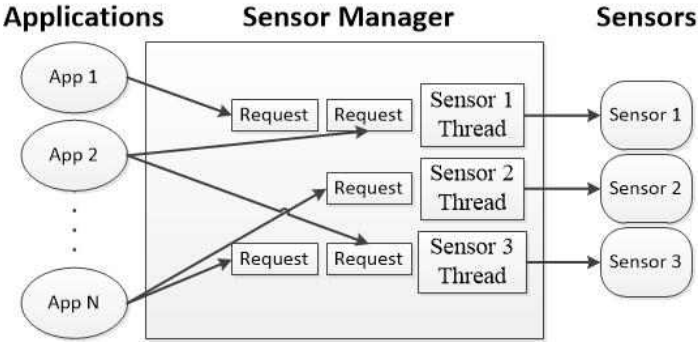

그림 3 제안하는 센서 프레임워크의 센서 수집 방식

3. 제안하는 센서 프레임워크

3.1 전체 센서 프레임워크 구조

본 논문에서 제안하는 센서 프레임워크는 자바스크립트 어플리케이션을 지원하기 위해 Node.js를 사용하고 어플리케이션과 센서 관리자는 서로 데이터를 주고받기 위해 dbus IPC(Inter-process Communication)를 사용한다. 또한 비동기 I/O를 위해 Node.js에서 사용하는 libuv 라이브러리를 사용하고 있으며, 센서 관리자 측에서 요청 대기를 위해 GMainloop를 사용하였다. 그림 2은 제안하는 프레임워크에 대한 전체적인 구조를 보여준다. 자바스크립트 어플리케이션과 센서 관리자 간에 인터페이스를 위해 Native Interface Layer (NIL)을 만들었으며, NIL은 자바스크립트 스레드와 Libuv 스레드로 구성되어 있다. 자바스크립트 스레드는 자바스크립트 어플리케이션과의 애드온 역할을 하며, Libuv 스레드는 센서 관리자로부터 dbus 메시지를 받는 역할을 한다. Libuv 스레드에서 받은 메시지를 자바스크립트 스레드에게 전달하고, 자바스크립트 스레드는 어플리케이션에게 콜백 함수를 통해 데이터를 전달한다.

그림 4는 센서 드라이버부터 어플리케이션까지의 센서 데이터의 흐름을 나타낸다. 센서 프레임워크는 센서 드라이버, 센서 관리자, NIL, 어플리케이션으로 총 네 가지 계층으로 되어 있다.

3.2 센서 드라이버

센서 드라이버는 다양한 형태의 센서 데이터를 습득하는 방식을 추상화하기 위한 계층이다. 다양한 센서들을 효율적으로 관리하기 위해 센서마다의 특징을 추상화하여 하나의 인터페이스로 제공한다. 그림 4에서 볼 수 있듯이, 센서 별로 이름, 타입 등 상태 정보와 수행 함수를 따로 정의할 수 있도록 센서를 모듈화 하였다. 센서를 모듈화 함으로써, 센서가 독립화되어 추가 센서를 사용하고자 할 때에, 정해진 특정 API에 대해서만 맞춰서 코드를 생성해주면 쉽게 사용할 수 있다. 예를 들어, 그림 4에서와 같이 센서 드라이버를 정의했다고 했을 때, 센서 드라이버는

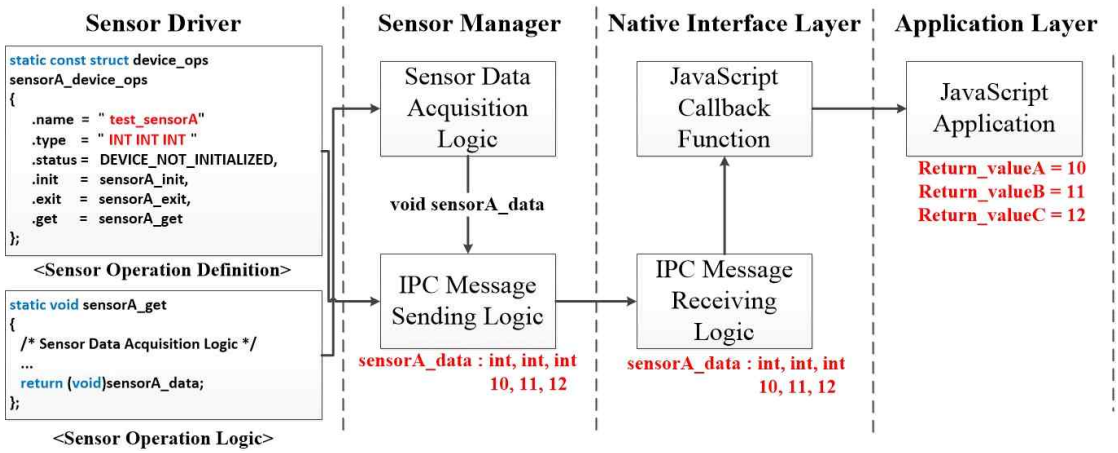

그림 4 센서 프레임워크 내 각 계층별 데이터 흐름

sensorA_init, sensorA_exit, sensorA_get 과 같은 함수를 통해 실행되며 해당 센서의 이름, 데이터의 타입, 상태도 다음과 같이 정의한다. 또한 해당 함수에 대한 정의를 그 아래에 센서 작동 로직에 따라서 센서 데이터를 수집한다.

3.3 센서 관리자

센서 관리자는 앞서 센서 드라이버를 통해 모듈화된 센서들을 일정한 로직을 통해서 센서 데이터를 수집하고 IPC를 통해 NIL로 전송한다. 그림 3은 센서 관리자의 센서 수집 방식에 대한 그림이다. 각 센서마다 하나의 스레드가 생성되며 센서가 데이터를 수집할 때가 아니면 모두 수면 상태에 있다. 주기적 수집과 이벤트 기반 수집 방식에 대한 요청이 오면, 해당 요청은 큐에 등록이 되어 해당 센서의 쓰레드는 깨어나 해당 요청이 끝날 때 까지 센서 데이터를 수집하여 전달한다. 각 쓰레드는 NIL 측에 있는 프로세스와 dbus를 통해 IPC 메시지 로직을 수행하며, 동기적, 비동기적 방식을 모두 지원한다. 센서 관리자는 이와 같이 센서마다 스레드를 생성하므로서, 여러 어플리케이션에서 같은 센서에 대한 데이터 정보를 요청할 때, 여러 스레드가 생성되지 않고 한 개의 해당 센서 스레드에서 센서의 데이터 정보를 어플리케이션에게 모두 제공한다. 그러므로 센서 데이터 요구 이벤트가 늘어나도 스레드 수는 일정하다.

3.4 Native Interface Layer (NIL)

NIL은 센서를 관리하기 위해 C언어로 작성된 센서 관리자와 자바스크립트로 구성된 어플리케이션 간에 결합과 서로 간의 효율적인 IPC를 위해 만들어진 자바스크립트 기반의 IPC 추상화 계층이다. 자바스크립트 어플리케이션을 사용하기 위해 자바스크립트용 서버향 플랫폼인 Node.js를 사용하고 있다. Node.js는 이벤트 기반의 논블로킹 I/O를 주 컨셉으로 만든 백엔드 플랫폼으로서, 하나의 스레드를 기반으로 프로그램이 실행되며, 함수들이 비동기적으로 작동한다.

NIL은 센서 관리자와 libuv와 dbus를 사용하여 동기적, 비동기적인 양방향 IPC를 지원한다. 비동기적으로 작동하기 위해서 libuv의 메인 루프와 스레드 풀을 사용한다. 상위 계층에서 실행되는 자바스크립트 코드와 NIL은 자바스크립트 콜백 함수와 요구 관리 부분에 의해서 연결되어 있고, Dbus를 사용해서 센서 관리자와 통신한다. 센서 관리자에게 메소드 콜을 보내면, 센서 관리자에서는 해당 콜에 대한 함수가 실행되고 그에 대한 답장 메시지를 다시 NIL 쪽으로 보낸다. NIL쪽에서는 해당 메시지를 IPC 메시지 수신 로직에 따라 받고, 해당 메시지를 콜백 함수를 통해 자바스크립트 콜백 함수 쪽으로 넘겨준다. 해당 자바스크립트 콜백 함수는 해당 데이터를 자바스크립트 코드쪽으로 리턴한다. 이를 통해 서비스 개발자의 자바스크립트 코드와 센서 관리자가 양방향 및 동기, 비동기적 통신을 할 수 있다.

4. 결론

사물인터넷 환경에서는 모바일 환경과는 달리 센서의 데이터 정보를 요구하는 어플리케이션 수가 많을 수 있다. 이 때 요구 이벤트 당 하나의 스레드를 할당하는 기존의 모바일 환경의 센서 프레임워크는 비효율적이며, 적합하지 않다. 그러므로 본 논문에서는 각 센서마다 스레드를 할당하는 방식의 센서 프레임워크를 제안한다.

참고 문헌

[1] *Programming Language Popularity Chart.* Retrieved May 14, 2016, from http://langpop.corger.nl
[2] *Android API document*, Retrieved May 14, 2016, from http://developer.android.com/intl/ko/reference/android/hardware/SensorManager.html
[3] *Tizen developer document*, Retrieved May 14, 2016, from https://wiki.tizen.org/wiki/Porting_Guide