논문 2015-10-17

IoT 서비스 플랫폼을 위한 리눅스 FUSE 기반 가상 파일 시스템

(A Virtual File System for IoT Service Platform Based on Linux FUSE)

이 형 봉, 정 태 윤*
(Hyung-Bong Lee, Tae-Yun Chung)

Abstract : The major components of IoT(Internet of Things) environment are IoT devices rather than the conventional desktop computers. One of the intrinsic characteristics of IoT devices is diversity in view of data type and data access method. In addition, IoT devices usually deal with real-time data. In order to use such IoT data for internal business or cloud services, an IoT platform capable of easy domain management and consistent data access interface is required. This paper proposes a Linux FUSE-based virtual file system connecting IoT devices on POSIX file system view. It is possible to manage IoT domain with the native Linux utilities such as mkdir, mknod, ls and find in the file system. Also, the file system makes it possible to access or control IoT devices through POSIX interface such as open(), read(), write() or close() without any separate APIs or utilities. A test result shows that the management performance of the file system is lower than that of linux file system negligibly.

Keywords : IoT, VFS, FUSE, USN, POSIX

Ⅰ. 서 론

IoT(Internet of Things)는 2000년대 초반 M2M (Machine-to-Machine), USN(Ubiquitous Sensor Network) 등의 이름으로 시작하여 현재 정부의 신성장 동력 산업 분야의 한 자리를 차지하고 있다[1]. IoT가 기존의 M2M이나 USN과 다른 점이 있다면 제각각 떨어져 존재하는 사물 간 통신을 거대한 하나의 틀로 묶어 서비스로 형상화시킴으로써 사람에게 좀 더 나은 부가가치를 주는 것 정도이다[2]. 이러한 IoT의 구현 및 활용을 위한 필요 기술로서 센싱 기술, 유무선 통신 및 네트워크 인프라 기술, IoT 서비스 인터페이스 기술 등 3 대 요소 기술을 들 수 있다[3]. 이들 중 센싱 기술과 유무선 통신 및 네트워크 인프라 기술은 4~5년 전 MEMS(Micro-Electro-Mechanical System) 기술과 함께 절정기를 맞았던 USN 산업에 의해 충분히 개발되었다. 그러나 USN 산업이 전망치와는 다르게 대규모 수요 창출 측면에서 한계에 직면하고 있는데, 그 이유 중의 하나로 USN을 기반으로 하는 서비스 부재가 지적되고 있다[4]. USN 토대 위에 서비스 개념이 도입되고 그 활용 영역이 일상생활 범주까지 확장되면 그 것이 곧 IoT 실현으로 볼 수 있다.

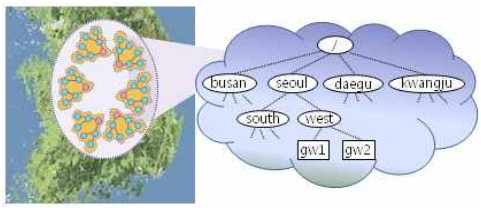

그림 1. IoT를 위한 가상 파일시스템 개념
Fig. 1 Concept of the Virtual File System for IoT

*Corresponding Author(tychung@gwnu.ac.kr)
Received: 22 Apr. 2015, Revised: 21 May 2015, Accepted: 30 May 2015.
H.-B. Lee, T.-Y. Chung: Gangneung-Wonju National Univ.
※ 이 논문은 2015년 미래창조과학부의 재원으로 SW융합기술고도화 사업의 지원을 받아 수행된 연구임 (S1005-14-1007).

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2015.10.3.139

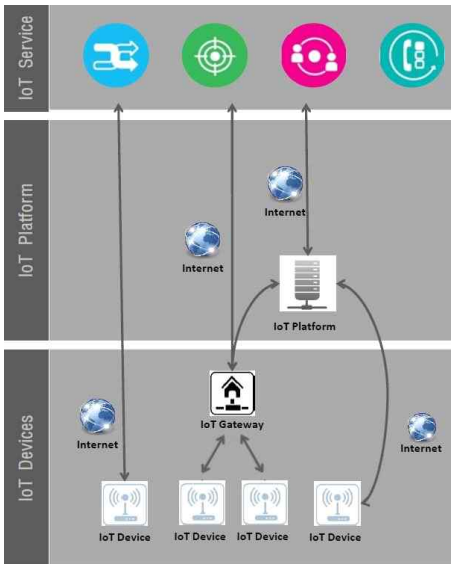

그림 2. IoT 서비스 참조 모델
Fig. 2 IoT Service Reference Model

USN에 서비스 개념을 도입하기 위해서는 개발자나 데이터 사용자에게 지리적으로 산재해 있는 다양한 IOT 디바이스들에 대한 통합적·계층적 탐색 뷰와 디바이스 각각의 이질성을 최대한 은폐하는 일관된 접근 인터페이스 제공이 무엇보다도 중요하다. 현재 대부분의 IoT 시스템들은 디바이스 명칭 관리를 위해 LDAP(Lightweight Directory Access Protocol)이나 Berkeley DB 등 별도의 툴을 사용하고 있으나[5], 이 연구에서는 그림 1과 같이 리눅스 파일 시스템 및 관련 명령어를 이용하여 IoT 디바이스를 관리하고 접근할 수 있는 리눅스 FUSE 기반 가상 파일 시스템을 구현한다. IoT 서비스 개발자나 데이터 사용자는 구현된 가상 파일시스템 환경에서 어떠한 새로운 API나 명령어를 사용하지 않고 IoT 디바이스 탐색과 데이터 접근이 가능하다.

이 논문은 2장 관련연구, 3장 IoT 가상 파일 시스템 구현, 4장 IoT 가상 파일 시스템 평가, 그리고 마지막 5장 결론으로 구성되어 있다.

II. 관련 연구

1. IoT 서비스 참조 구조

IoT 서비스로부터의 디바이스 접근 경로는 그림 2와 같이 직접 통신, IoT 게이트웨이를 통한 통신, IoT 플랫폼을 통한 통신 등 크게 세 가지로 분류할

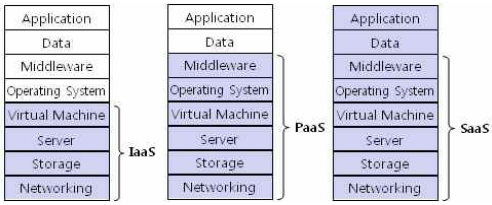

그림 3. 클라우드 서비스의 분류
Fig. 3 Classification of Cloud Service

수 있다[6]. 이 그림에서 IoT 게이트웨이는 주변에 위치하는 다양한 종류의 IoT 디바이스들로부터 시리얼 통신 등 로컬 통신을 통해 데이터를 수집하는 데이터 로거 역할을 담당한다. 최근 IoT 관련 업계에서는 이질적인 디바이스 데이터에 대한 일관된 데이터 뷰를 위한 표준을 선점하고, 기존의 USN 기술과의 연계를 위해 IoT 게이트웨이 개발에 치중하고 있다[7].

IoT 플랫폼은 IoT 게이트웨이나 디바이스와 통신하여 하부의 모든 IoT 구성 요소들에 대한 통합 뷰와 데이터 접근 인터페이스를 제공함으로써 IoT 서비스가 운영될 수 있는 운영체제 성격의 플랫폼을 제공한다. 이는 그림 3에 보인 클라우드 서비스 유형[8] 중 PaaS(Platform as a Service) 개념과 유사한데, 다만 구성 요소가 단순한 파일 형태뿐만 아니라 센서나 엑추에이터 등 생명체적 개체가 주류를 이루기 때문에 실시간적 양방향 데이터 흐름이 중요시된다는 점이다. IoT 플랫폼의 가장 중요한 역할은 IoT 서비스를 개발하고자 하는 기업이나 개인에게 그들이 설치하여 운용 중인 디바이스들에 대한 명칭 관리 및 접근 인터페이스를 편리하게 플러그인할 수 있는 틀을 제공하는 일인데, 이는 아래와 같이 크게 네 가지 모듈로 세분화될 수 있다[9].

□ D-플랫폼(Device-Platform)

디바이스 플랫폼은 다양한 IoT 디바이스(사물)를 IoT 인프라 틀에 연결하여 IoT 서비스를 위해 가장 기본적으로 필요한 디바이스 접근 인터페이스를 제공한다. 플랫폼 개발자는 서비스 개발자나 사용자들에게 고유의 API를 제공하여 IoT 데이터에 접근할 수 있도록 한다.

□ P-플랫폼(Planet-Platform)

플래닛 디바이스는 IoT 글로벌 환경에서 디바이스의 등록 및 검색을 지원하는 모듈로서 디바이스의 고유 이름, 설치 위치 등을 관리한다. 전형적인

구현 방법 중의 하나로 인터넷 환경에서 계정 관리 등에 널리 활용되는 LDAP을 들 수 있다.

□ M-플랫폼(Mashup-Platform)

매쉬업 플랫폼은 IoT 데이터들 사이의 시계열 통계 정보나 상관성에 따라 관련 정보를 융합하여 사용자에게 보다 유용한 정보가 제공될 수 있도록 가공 정보를 생성하고 관리한다.

□ S-플랫폼(Store-Platform)

스토어 플랫폼은 개방형 IoT 디바이스에 탑재할 수 있는 디바이스 어플리케이션, IoT 플랫폼을 통해 IoT 데이터에 접근할 수 있는 API, 그리고 완성된 IoT 서비스 어플리케이션 등을 관리하여 사용자가 원하는 서비스를 검색하여 다운로드 할 수 있도록 관리한다.

2. IoT 디바이스 및 게이트웨이

현재 가장 활발하게 개발되고 있는 IoT 시스템 유형은 개인이 휴대하거나 가정에 설치된 단일 품목의 IoT 디바이스 및 게이트웨이가 스마트폰과 직접 통신하거나 중앙의 단순 서버를 통해 스마트폰에 데이터를 공급하는 모델이다. 이를테면 위딩스 혈압계(Withings Blood Pressure Monitor)[9, 10]는 휴대형 혈압계로부터의 측정 결과를 주변의 스마트폰으로 즉시 전송하여 확인 가능하고, 서버에 보관하여 지속적인 건강관리에 활용한다. 벨킨 위모(Belkin WEMO)[9, 11]는 WiFi 및 3G/4G로 구성된 홈 네트웨크 환경에서 전원 콘센트에 부착되어 스마트폰과의 직접 통신으로 전원 on/ff는 물론 전원 사용 현황을 모니터링 한다. 글로우캡(GlowCap)[9, 12]은 약병 뚜껑에 여닫이 활동을 감지하는 디바이스가 데이터를 무선으로 벽부 전원부에 설치된 게이트웨이로 보내면 게이트웨이는 3G 통신으로 서버에 전달한다. 서버는 간호사 등 해당 데이터에 대한 수신 예약자들에게 SMS나 이메일 등으로 통보하여 환자 관리에 활용된다. 아두이노(Arduino)[13]는 이와 같은 IoT 디바이스 및 게이트웨이 개발 플랫폼으로 부상했고, MQTT(Message Queue Telemetry Transport)[14]는 디바이스들로부터 수집된 데이터를 서버로 전송하는 IoT 게이트웨이 역할이 주요 목적이지만, 수집된 데이터에 대한 도메인 기반 웹 서버 기능도 가능하다. 위 사례들의 공통점은 다양한 고객 및 다양한 IoT 디바이스를 통합적으로 관리·운용하는 큰 틀의 IoT 클라우드 플랫폼에 대한 지원은 충분하지 못하다는 점이다.

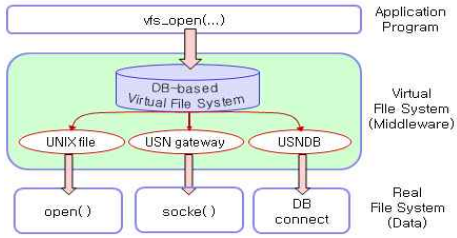

그림 4. 가상 파일시스템을 이용한 IoT 플랫폼
Fig. 4 IoT Platform using a Virtual File System

3. IoT 플랫폼

본격적인 IoT 서비스 플랫폼의 개발이나 제안은 많지 않아 주목할 만한 사례가 겨우 몇 가지에 불과하다. 모비우스 플랫폼(MOBIUS Platform)[9, 15]은 국내 KETI와 SKT가 개발하여 현재 상용 서비스 중에 있는 IoT 플랫폼이다. 이 서비스 환경에서는 고객(사업자)이 서비스 사이트에 가입하여 원하는 디바이스 및 접근 API를 등록해두고, 일반인을 대상으로 해당 디바이스, API, 어플리케이션 등을 판매·배포하도록 한다. 이를 지원하기 위하여 디바이스 플랫폼을 &CUBE라 명명하고 그 세부 유형을 게이트웨이 역할을 하는 Rosemary, 독립적인 IoT 디바이스로 활동하는 Lavender 등으로 분류하여 다양한 형태의 디바이스 수용이 용이하도록 한다. 본 연구의 목적과 가깝고 보다 보편적인 IoT 서비스 플랫폼으로 Xively[16]를 들 수 있다. Xively는 데이터 소스를 사이트에 등록하고, 등록된 데이터를 다른 사람과 공유한다는 점에서 유튜브와 유사한데, 공유 데이터가 주로 센서에서 생산되는 실시간 환경 모니터링 데이터라는 점이 다르다.

창의적 아이디어에 의한 범 산업적 IoT 서비스 수요를 창출하기 위해서는 서비스 기획 및 개발자가 IoT 디바이스를 저렴한 비용으로 편리하게 관리하고 접근할 수 있는 개방형 작업장(Workbench)이 주어져야 한다. 즉, 서비스 개발의 기반이 되는 개방형 플랫폼 개발이 필수적이지만 아직까지는 그 개념 인식조차 미흡하여 빅데이터 플랫폼과 IoT 플랫폼을 혼용하는 경우도 있고, 개발되었다 해도 폐쇄적이어서 내부 구조에 대해서는 알려진 바가 거의 없다. 이런 현실을 극복하기 위해 IoT 디바이스들을 DB 기반 가상 유닉스 파일 시스템으로 관리하는 방안[17]이 제안되었다. 이 플랫폼에서는 디바이스의 등록과 검색을 위해 mkdir, mknod, ls, find 등 유닉스 명령어와 동일한 사용법의 유틸리티를 사용하고, 디바이스 접근을 위해서는 open(), read(),

write(), ioctl(), close() 등 POSIX(Portable Operating System Interface)[18] 표준을 사용함으로써 서비스 개발자에게 편리하고 친근한 개발환경을 제공하고자 하였다. 그러나 이 제안은 그림 4와 같이 오로지 사용자 영역에서 구현되므로 가상 파일시스템 접근을 위한 미들웨어와 이를 참조하는 유틸리티 모두를 새롭게 개발해야 한다는 단점이 있다. 뿐만 아니라, 입·출력에서도 POSIX 인터페이스를 완벽하게 구현하는 데 한계가 있다. 이 문제를 해결하기 위해 본 논문에서는 리눅스의 FUSE(File System in Userspace)[19]를 이용하여 가상 파일 시스템을 리눅스 파일 시스템 후면에 구현함으로써 탑재된 POSIX 유틸리티와 인터페이스를 그대로 활용할 수 있도록 한다. 또한 가상 파일시스템 구현에 DB를 사용하지 않고 리눅스 파일 시스템을 활용하여 성능과 안정성을 높인다.

Ⅲ. IoT 플랫폼을 위한 리눅스 FUSE 기반 가상 파일시스템 구현

1. 리눅스 FUSE와 IoT 가상 파일시스템

□ FUSE 메커니즘

FUSE는 그림 5과 같이 사용자 수준에서 구현되어 마운트된 파일시스템을 POSIX 인터페이스로 참조하면 커널을 거쳐 사용자 영역에서 실행 중인 가상 파일시스템 디몬으로 제어가 넘어간다. 즉, FUSE는 리눅스 파일시스템 접근을 위해 구현된 프리미티브 프로시저들의 집합이고, 이들 프로시저들은 FUSE 디몬에 의해 호출되어 사용자가 의도하는 파일시스템 뷰를 제공한다. 그림 6에 FUSE 파일시스템 구현 프레임을 보였는데, fuse_main()이 FUSE 디몬이고 마운트 포인트와 프로시저 집합을 인수로 받는다.

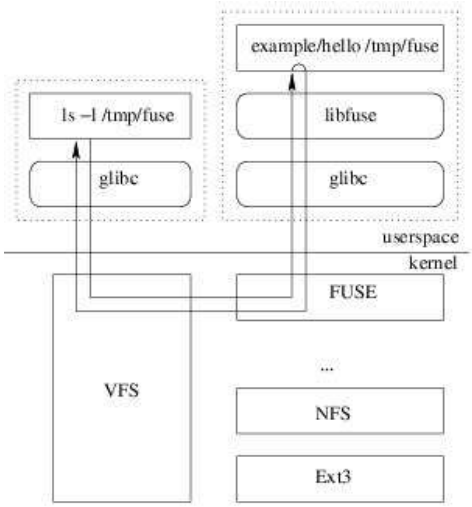

그림 5. 리눅스 FUSE 개념
Fig. 5 Concept of the Linux FUSE

```
static struct fuse_operations iotfs_oper = {
    .mkdir  = iotfs_mkdir,
    .mknod = iotfs_mknod,
    .getattr = iotfs_stat,
    .readdir= iotfs_readdir,
    .open   = iotfs_open,
    .read   = iotfs_read,
    .write  = iotfs_write,
    .ioctl  = iotfs_ioctl,
    .rmdir  = iotfs_rmdir,
    .unlink = iotfs_unlink,
    .flush  = iotfs_flush,
    .release= iotfs_close,
};
int main(int argc, char *argv[])
{   fuse_main(argc, argv, &iotfs_oper, NULL); }
```

그림 6. FUSE 구현 프레임워크
Fig. 6 Framework of FUSE Implementation

□ IoT 가상 파일 시스템

POSIX 파일시스템은 파일의 유형을 크게 일반 파일, 디렉터리 파일, 디바이스 파일 등 세 가지로 분류한다. 일반 파일은 파일시스템 내에 저장된 데이터를 의미하므로 이 파일에 대한 접근은 파일 시스템 내부에서 처리된다. 디렉터리 파일은 일반 파일과 동일하나 다만 이 파일의 내용(데이터)이 파일시스템을 조직하기 위한 메타 정보이므로 사용자는 그 내용을 직접 편집할 수 없고 운영체제(파일시스템)에 의뢰하여 생성하거나 수정할 수 있다. 디바이스 파일은 저장 내용은 없고, 속성으로 설정된 디바이스 번호를 참조하여 해당 입·출력 프로시저를 연결하는 가교 역할만 담당한다.

이 연구에서 추구하는 기본 방법은 다양한 타입의 IoT 디바이스를 POSIX 디바이스 파일로 대응시키고, 각 타입별 접근 방법을 디바이스 드라이버 개념으로 플러그인 하는 것이다. 이렇게 하면 IoT 디바이스에 접근하고자 하는 모든 사용자는 POSIX 인터페이스로 IoT 디바이스에 접근할 수 있을 뿐 아니라, 파일시스템이 제공하는 디렉터리 계층성에 의해 도메인 관리 및 검색이 자연스럽게 이루어진

```
static int iotfs_mkdir(const char *path, mode_t mode)
{
    char    buf[256];
    sprintf(buf, "%s/%s", VFS_ROOT, path);
    return mkdir(buf, mode);
}
```

그림 7. Iotfs_mkdir() 프로시저
Fig. 7 Iotfs_mkdir() Procedure

다. 그런데 이 방법을 커널 영역에 적용하는 일은 현실적으로 불가능하다. 이를 해결하는 적절한 수단이 리눅스 FUSE 메커니즘을 활용하는 가상 파일 시스템이다.

2. IoT 가상 파일시스템 설계

□ IoT 디바이스와 가상 파일시스템 노드 대응

IoT 디바이스(혹은 IoT 게이트웨이)를 식별하는 필수 요소는 사용자가 부여하는 해당 디바이스의 명칭, 타입, 그리고 IP 주소 등 세 가지이다. 이 세 가지 식별 요소를 가상 파일 시스템의 노드(파일)로 대응시켜, 특정 노드에 대한 open(), read() 등의 입·출력 접근이 인터넷을 통해 대응되어 있는 IoT 디바이스로 연결되도록 한다. 이를테면 파일명 “seoul_usngw g 12.23.34.45”은 IP 주소가 12.23.34.45인 서울의 USN 게이트웨이를 의미한다.

□ 파일시스템 노드 구성

위에 보인 파일명의 예에서, 디바이스 명칭과 타입, IP 주소 모두를 파일명에 포함시키면 파일명이 길어져서 디바이스 접근이나 탐색 시 불편하므로, 디바이스 명칭만을 파일명으로 사용하고 나머지 타입과 주소는 속성 정보로 관리하는 것이 좋다. 그런데, POSIX 파일시스템의 디바이스 파일은 디바이스 타입으로 문자 디바이스('c': S_IFCHR), 블록 디바이스('b': S_IFBLK), 파이프 디바이스('p': S_IFFIFO) 외에는 허용하지 않을 뿐 아니라, 속성 정보도 디바이스 번호 외에 추가로 설정하는 방법이 없다. 따라서 이 연구에서는 디바이스 파일 대신 일반 파일을 사용하되 IoT 디바이스 명칭을 파일명으로 사용하고, 타입과 주소 등 기타 속성 정보는 파일 내용으로 저장하도록 한다.

□ IoT 디바이스 접근 프로시저

이 논문에서는 수동형 IoT 디바이스와 능동형 IoT 디바이스에 대한 전형적인 접근 프로시저 템플릿을 제시한다. 여기서 수동형이란 IoT 플랫폼으로

```
static int iotfs_mknod(const char *path, mode_t
                                    mode, dev_t rdev)
{
    (void)rdev;
    int     rc;
    char    buf[64], type[2], iotfd[32];
    if (get_path_type_iot(path, type, iotfd) < 0)
        return (-EINVAL);
    sprintf(buf, "%s/%s", VFS_ROOT, path);
    mode &= ~(S_IFIFO | S_IFCHR | S_IFBLK);
    mode |= S_IFREG;
    if ((fd = creat(path, mode)) < 0)
        return -errno;
    sprintf(buf, "[%c]%s", type[0], iotfd);
    write(fd, buf, strlen(buf));
    close(fd);
    return(0);
}
```

그림 8. Iotfs_mknod() 프로시저
Fig. 8 Iotfs_mknod() Procedure

부터 연결 요청을 받아 응답하는 디바이스를, 능동형이란 IoT 플랫폼에 연결을 요청하여 통신하는 디바이스를 말한다. 3G/4G 등의 이동 통신을 통해 인터넷에 접속하거나 DHCP로 IP를 할당받는 등의 경우는 능동형 디바이스가 되어야 한다.

3. IoT 가상 파일시스템 구현

□ 가상 파일시스템의 물리적 공간

FUSE 기반 IoT 가상 파일시스템 조직을 위한 물리적 공간으로 DB를 사용할 수 도 있으나, 이 연구에서는 성능과 안정성을 위해 리눅스 파일시스템의 일부(예: “/IOT_VFS”)를 사용한다. 만약 사용자가 FUSE 가상 파일시스템을 “/home/iotfs”에 마운트했다면 사용자는 “/home/iotfs” 이하에 IoT 가상 파일시스템을 운용하고, FUSE 내부에서는 그 내용을 “/IOT_VFS” 이하에 관리한다.

□ 파일 시스템 디렉터리 생성(mkdir())

“mkdir /home/iotfs/seoul”와 같이 mkdir 명령어를 사용하면, mkdir() 시스템 콜을 거쳐 FUSE 가상 파일시스템의 iotfs_mkdir() 프로시저에 경로명 “/seoul”과 기본 모드가 전달된다. iotfs_mkdir()에서는 주어진 경로명을 내부 공간의 경로명인 “/IOT_VFS/seoul”로 변환하고 mkdir() 시스템 콜을 사용하여 디렉터리를 생성한다(그림 7).

```
static int iotfs_open(const char *path, struct
                              fuse_file_info *fi)
{
     char     buf[256], type[2], iotfd[32];
     int      ffd =-1, fd, n;
     mode_t mode;
     struct   stat st;
     sprintf(buf, "%s/%s", VFS_ROOT, path +1);
     if (stat(buf, &st) < 0) return -ENOENT;
     if (st.st_mode & S_IFDIR) return -EINVAL;
     fd = open(buf, O_RDONLY);
     n = read(fd, buf, sizeof(iotfd)); buf[n] = 0;
     close (fd);
     strcpy(iotfd, buf + 3);  mode = st.st_mode;
     type[0] = buf[1];
     if ((ffd = vfs_open_iot(type[0], iotfd,
                        fi->flags & O_APPEND)) < 0)
        return ffd;
     fi->fh = ffd; fi->direct_io = 1;
     fi->nonseekable = 1;
     return 0;
}
```

그림 9. Iotfs_open() 프로시저 1
Fig. 9 Iotfs_open() Procedure 1

□ IoT 노드 생성(mknod())

mkdir과 유사하게 “mknod /home/iotfs/seoul/usngw:g:12.23.34.45:67 p”와 같이 파이프 디바이스 생성 명령어를 사용한다. 이 때, 파일명에 공백을 주지 않기 위해 “파일명:타입:주소:포트” 형태의 파일명을 사용하도록 한다. mknod 명령어 인수 내용은 mknod() 시스템 콜을 거쳐 FUSE 가상 파일 시스템의 iotfs_mknod() 프로시저에 경로명 “/seoul/usngw:g:12.23.34.45:67”과 디바이스 타입 'p'로 전달된다. iotfs_mknod()는 경로명을 분석하여 명칭 부분(“/seoul/usngw”)과 속성 부분(“:g:12.23.34.45:67”)을 분리한 후, 파이프 타입 대신 creat() 시스템 콜을 이용하여 내부 공간에 “/IOT_VFS/seoul/usngw” 파일을 생성하고 그 파일 내용으로 속성 부분을 저장한다(그림 8).

□ IoT 노드 개방(open())

사용자가 “fd = open(”/home/iotfs/seoul/usngw”, O_RDWR)”와 같이 open() 시스템 콜을 시도하면, 경로명 “/seoul/usngw”와 개방 모드 O_RDWR 값이 FUSE 가상 파일 시스템의 iotfs_open() 프로시저에 전달된다. iotfs_open() 프로시저(그림 9, 10)는 내부 공간 경로명으로 변환한

```
int vfs_open_iot(char type, char *iotfd, int flag)
{
    int     fd, sd, rc;
    struct  sockaddr_in  srv_addr;
    if (get_addr_port(iotfd, &srv_addr) < 0)
       return -EINVAL;
    switch(type) {
    case 'g' :
      sd = socket(AF_INET, SOCK_STREAM, 0);
      rc = connect(sd, &srv_addr, sizeof(srv_addr));
      if (rc < 0) { close(sd); return rc; }
      flag = TYPE_SOCK;
      break;
    case 's' :
      if (flag) {                  // try listen
        sd = socket(AF_INET, SOCK_STREAM, 0);
      if (bind(sd, &srv_addr, sizeof(srv_addr))<0)
          { close(sd); return (-errno); }
        listen(sd, 1);
        if (put_lstn_port(srv_addr.sin_port, sd)<0)
          { close(sd); return -ENOMEM; }
        flag = TYPE_LSTN;
      } else {
        fd_set  rdfds;
        struct  timeval  to;
        if ((fd=get_lstn_port(srv_addr.sin_port))<0)
           return (-EINVAL);
        FD_ZERO(&rdfds); FD_SET(fd, &rdfds);
        to.tv_usec = 0; to.tv_sec = MAX_TOUT;
        if (select(fd+1, &rdfds, NULL, NULL,
                &to) <= 0) return -EAGAIN;
        if ((sd = accept(fd, NULL, NULL)) < 0)
           return (-errno);
        flag = TYPE_SOCK;
    }
    fd = vfs_get_fd();
    Ufile[fd].flag = flag;
    Ufile[fd].data = (char *)sd;
    Ufile[fd].size = 0;
    return fd;
}
```

그림 10. Iotfs_open() 프로시저 2
Fig. 10 Iotfs_open() Procedure 2

“/IOT_VFS/seoul/usngw” 파일을 개방하여, mknod 명령어에서 저장해 두었던 속성정보(타입,주소,포트)를 읽는다. 타입이 수동형 디바이스(‘g’)인 경우 소켓 “sd = connect()”를 수행하고, 능동형 디바이스(‘s’)인 경우에는 소켓 listen() 후 “sd = accept()”를 수행하되, 연결 요청이 없더라도 디바이스 행업(hang up)을 예방하기 위한 방안이 필요하다. 이를 위해 사용자에게 2 단계 개방 절차를 제시한다. 개

```
static int iotfs_read(const char *path, char *buf,
             size_t size, off_t offset,
                      struct fuse_file_info *fi)
{
     (void)offset;
     return vfs_read(fi->fh, buf, size, offset);
}
static int iotfs_write(const char *path,
       const char *buf, size_t size,
            off_t offset, struct fuse_file_info *fi)
{
     (int)offset;
     return vfs_write(fi->fh, buf, size, offset);
}
int vfs_read(int fd, char *buf, int  size, int offset)
{
    UFILE  *fp;
    if (fd < 0 || fd >= MAX_FILE ||
                        !(fp = &Ufile[fd])->flag)
       return -EINVAL;
    if (fp->flag & TYPE_SOCK)
       return recv((int)fp->data, buf, size));
   else
       return -EINVAL;
}
int vfs_write(int fd, char *buf, int  size, int offset)
{
    UFILE  *fp;
    if (fd < 0 || fd >= MAX_FILE ||
                        !(fp = &Ufile[fd])->flag)
       return -EINVAL;
    if (fp->flag & TYPE_SOCK)
       return send((int)fp->data, buf, size));
   else
       return -EINVAL;
}
```

그림 11. Iotfs_open()/itfs_write() 프로시저
Fig. 11 Iotfs_open()/iotfs_write Procedure

방 모드에 O_APPEND 플래그를 설정하면 1 단계 개방을 의미하는데 이는 소켓의 listen()까지의 절차를 진행하고, sd와 함께 포트를 기록해 두어 2 단계 개방에서 참조될 수 있도록 한다. O_APPEND 플래그가 없는 2 단계 개방에서는 “sd = accept()”를 수행하는데 이 때 select()를 활용하여 일정 기간(10초) 후 EAGAIN 에러를 리턴할 수 있는 비봉쇄 입·출력 방식으로 동작하도록 한다.

“sd = connect()”나 “sd = accept()”에 성공하면 파일 테이블 하나를 할당하여 그 곳에 sd를 기록한 후 해당 파일 테이블의 번호를 open() 시스템 콜의 결과 값으로 알려준다.

```
static int iotfs_close(const char *path,
                       struct fuse_file_info *fi)
{
    (void)path; (void)fi;
    return vfs_close(fi->fh);
}
int vfs_close(int fd)
{
    UFILE  *fp;
    if (fd < 0 || fd >= MAX_FILE ||
                       !(fp = &Ufile[fd])->flag)
       return -EINVAL;
    if (fp->flag & TYPE_SOCK)
       close((int)fp->data);
    else if (fp->flag & TYPE_LSTN) {
       close((int)fp->data);
       del_lstn_port((int)fp->data);
    }
    else if (fp->flag & TYPE_FILE)
       close((int)fp->data);
    else if (fp->data) free(fp->data);
       fp->flag = 0;
    return(0);
}
```

그림 12. Iotfs_close() 프로시저
Fig. 12 Iotfs_close() Procedure

□ IoT 노드 입·출력(raed()/write())

“n = read(fd, buf, sizeof(buf))”나 “n = write(fd, buf, sizeof(buf))”에 의한 입·출력 시스템 콜은 FUSE 가상 파일 시스템의 iotfs_read()와 iotfs_write() 프로시저로 각각 연계되고, 그곳에서 소켓 recv()와 send()가 이루어진다(그림 11).

□ IoT 노드 폐쇄(close())

Close() 시스템 콜은 FUSE 가상 파일 시스템의 iotfs_release() 프로시저에 연결되어 해당 소켓을 폐쇄하고 할당되었던 파일 테이블을 반납한다(그림 12). 기타 디렉터리 삭제(rmdir()), IoT 노드(파일) 삭제(unink), 노드 속성 얻기(stat()), 디렉터리 읽기(readdir()), 속성 설정(ioctl) 등은 POSIX 인터페이스를 참조하여 구현하였다.

Ⅳ. FUSE 기반 가상 파일시스템 평가

구현된 FUSE 가상 파일시스템의 타당성을 검증하기 위해 설계 기능을 시험하고 노드 생성 및 검색 성능을 평가한다.

```
/home/hblee> ./iotvfs /home/iotfs
/home/hblee> mount | grep iotvfs
iotvfs on /home/iotfs type fuse.iotvfs (rw,nosuid,nodev,user=hblee)
/home/hblee> ls -l /home/iotfs
합계 0
/home/hblee> mkdir /home/iotfs/seoul
/home/hblee> mknod /home/iotfs/seoul/usngw:g:114.71.70.61:7777 p
/home/hblee> mknod /home/iotfs/seoul/usnsv:s:0.0.0.0:7777 p
/home/hblee> ls -lR /home/iotfs
/home/iotfs:
합계 4
drwxr-xr-x 2 hblee hblee 4096 2015-03-11 15:05 seoul

/home/iotfs/seoul:
합계 8
-rw-r--r-- 1 hblee hblee 20 2015-03-11 15:04 usngw [g]114.71.70.61:7777
-rw-r--r-- 1 hblee hblee 15 2015-03-11 15:05 usnsv [s]0.0.0.0:7777
/home/hblee>
```

그림 13. 디렉터리와 노드(그림 14)를 생성하는 과정
Fig. 13 Process of Directory and Node Creation for Fig. 14

표 1. 실험 환경
Table 1. Experimental Environment

Items		Specification/Version
Host	O.S.	Window 7, 32Bit
	CPU	Intel Quad Q9300, 2.5GHz
	Mem	2.00GB
VMware	Ver.	3.1.3 build-324285
Liniux		Ubunto 10.10
FUSE		2.9.3
MySQL		5.5.15-1

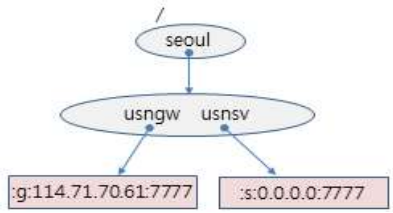

그림 14. 시험을 위한 IoT 가상 파일시스템
Fig. 14 IoT Virtual Filesytem for Test

그림 15. 수동형 IoT 디바이스 접근 시험
Fig. 15 Test for a Passive IoT Device

```
struct sockaddr_in  Addr; // dev_passive
main()
{
    int     sd, nsd, t, h, i;

    sd = socket(AF_INET, SOCK_STREAM, 0);
    Addr.sin_family = AF_INET;
    Addr.sin_port = htons(7777);

    bind(sd, &Addr, sizeof(Addr));
    listen(sd, 1);

    while(1) {
       nsd = accept(sd, NULL, NULL);
       printf("Accept ...₩n");
       for (i = 0; i < 5; i++) {
          t = rand() % 10, h = rand() % 10 + 60;
          sprintf(buf, "T:%.2d,H:%2d₩n", t, h);
          send(nsd, buf, F_SIZE, 0);
          printf("send() %s", buf);
       }
       printf("Finish ...₩n"); close(nsd);
    }
}
```

그림 16. 수동형 IoT 디바이스(게이트웨이)
Fig. 16 A Passive IoT Device(Gateway)

1. 시험 및 평가 환경

기능 시험 및 성능 평가는 윈도우 7 VMware 내에 설치된 리눅스 환경에서 진행되었고, 구체적인 세부 사양은 표 1과 같다.

2. FUSE 가상 파일시스템 기능 검증

□ 디렉터리·노드 생성 및 검색 기능

리눅스 고유 명령어 mknod와 ls를 사용하여 그림 14의 IoT 가상 파일시스템을 구성하고 검색하

```
main(int ac, char *av[])  // app for passive IoT dev
{
    int     fd, n, len;

    fd = open(av[1], O_RDWR);

    printf("Open ...₩n", fd);
    while(1) {
       for (len = 0; len < F_SIZE; len += n) {
          if ((n = read(fd, buf + len,
                        F_SIZE - len)) <= 0)
             printf("Finish ...₩n"), exit(0);
       }
       printf("read() %s", buf);
    }
    close(fd), exit(0);
}
```

그림 17. 수동형 IoT 디바이스 접근 어플리케이션
Fig. 17 An Application for Passive IoT Devices

```
struct  sockaddr_in    Addr;
char    buf[16];
main()
{
    int       sd, len, h, t, i;

    sd = socket(AF_INET, SOCK_STREAM, 0);

    Addr.sin_family = AF_INET;
    Addr.sin_port = htons(7777);
    Addr.sin_addr.s_addr = inet_addr("114.71.70.69");

    if (connect(sd, &Addr, sizeof(Addr)) < 0)
       { sleep(3); continue; }

    printf("Connect ...₩n");

    for (i = 0; i < 5; i++) {
          t = rand() % 10; h = rand() % 10 + 60;
          sprintf(buf, "T:%.2d,H:%2d₩n", t, h);
          send(nsd, buf, F_SIZE, 0);
          printf("send() %s", buf);
    }
    close(sd);
    printf("Finish ...₩n");
}
```

그림 18. 능동형 IoT 디바이스(게이트웨이)
Fig. 18 An Active IoT Device(Gateway)

는 과정을 그림 13에 보였다. 이들 그림에서 usngw는 수동형 디바이스 이고, usnsv는 능동형 디바이스이다.

□ 수동형 IoT 디바이스 접근 기능

그림 15에 수동형 IoT 디바이스(그림 16)에 대한 IoT 어플리케이션(그림 17)과 cat 유틸리티의 접근 모습을 보였다. 특히 cat과 같은 유틸리티에 의한 직접 접근은 IoT 디바이스 개발 과정에서 매우 유용하게 활용될 수 있다.

```
algo:/>dev_active
Connect ...
send() T:08,H:68
send() T:03,H:65
send() T:01,H:67
send() T:00,H:69
send() T:02,H:66
Finish ...
```

```
/> app /home/iotfs/seoul/usnsv
Accept ...
read() T:08,H:68
read() T:03,H:65
read() T:01,H:67
read() T:00,H:69
read() T:02,H:66
Finish ...
```

그림 19. 능동형 IoT 디바이스 접근 시험
Fig. 19 Test for an Active IoT Device

```
char  buf[16];
main(int ac, char *av[])
{
    int     ld, fd, len, n;

    ld = open(av[1], O_WRONLY | O_APPEND);

    while(1) {
       if ((fd = open(av[1], O_RDWR)) >= 0)
          break;
       if (errno != EAGAIN) exit(1); // critical !!!
       sleep(3);
    }
    printf("Accept ...₩n"); close(ld);
    while(1) {
       for (len = 0; len < F_SIZE; len += n) {
         n = read(fd, buf + len, F_SIZE - len);
         if (n <= 0)
             printf("Finish ...₩n"), exit(0);
       }
       printf("read() %s", buf);
    }
    close(fd);
}
```

그림 20. 능동형 IoT 디바이스(게이트웨이)
Fig. 20 An Active IoT Device(Gateway)

```
/home/hblee> find /home/iotfs
/home/iotfs
/home/iotfs/seoul
/home/iotfs/seoul/usnsv [s]0.0.0.0:7777
/home/iotfs/seoul/usngw [g]114.71.70.61:7777
/home/hblee> rm -rf /home/iotfs/*
/home/hblee> find /home/iotfs
/home/iotfs
/home/hblee>
```

그림 21. 디렉터리와 노드 삭제 과정
Fig. 21 Process of Directory and Node Removal

□ 능동형 IoT 디바이스 접근 기능

능동형 IoT 디바이스(그림 18)에 대한 IoT 어플리케이션(그림 20)의 접근 모습을 그림 19에 보였다.

□ 디렉터리・노드 삭제 기능

그림 13에서 생성한 가상 파일시스템의 디렉터리와 파일들을 find 유틸리티로 검색한 다음, 이들을 rm 명령어로 삭제하는 과정을 그림 21에 보였다.

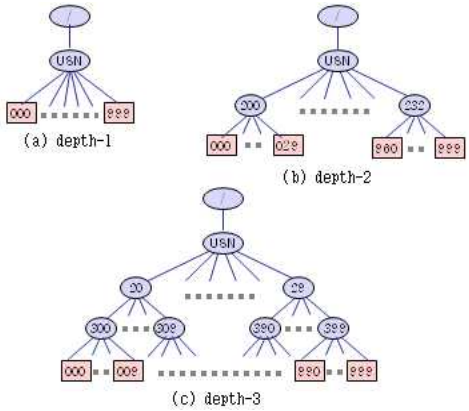

그림 22. 성능 평가 환경
Fig. 22 Performance Test Environment

3. FUSE 가상 파일시스템 성능 검증

디렉터리와 노드를 생성하고 개방하는 과정에서의 시간적 성능을 평가하기 위해 1000개의 노드를 그림 22과 같이 1단(depth-1), 2단(depth-2), 3단(depth-3) 등 세 가지의 디렉터리 체계로 생성하고 개방하는 시간(순수한 CPU 실행 시간이 아닌 대기 시간 포함)을 리눅스 파일시스템과 IOT-VFS(구현 가상 파일시스템)에서 측정하였다. 주어진 분석 데이터는 10회의 생성과 탐색 시간이고, 이에 대한 5회 측정치의 평균이다.

□ 디렉터리·노드 생성 시간

표 2에 리눅스 고유 파일시스템과 구현된 가상 파일시스템의 디렉터리 및 노드의 생성시간을 비교해 보였는데, 가상 파일시스템의 생성시간이 리눅스 파일시스템의 약 11배이다.

□ 노드 개방 시간

표 3에는 두 파일시스템의 노드 개방시간을 비교해 보였는데, 가상 파일시스템의 개방시간이 리눅스 파일시스템의 약 2.5배이다.

□ 측정 결과 분석

구현 가상파일 시스템은 리눅스 고유 파일시스템을 경유하여 후면의 사용자 영역에서 어플리케이션으로 구현되므로 상대적인 성능이 저조한 것으로 분석된다. 생성 시간의 경우 큰 차이를 보이고 있는데, 디렉터리나 노드의 생성 빈도가 높지 않은 전형적인 읽기 위주의 환경이므로 크게 문제되지 않는다. 개방 시간은 격차가 크게 줄어 근소한 차

표 2. 디렉터리와 노드 생성 시간(ms)
Table 2. Directory and Node Creation Time

File system	depth-1	depth-2	depth-3
Linux	306	206	201
IOT-VFS	2623	2560	2520

표 3. 노드 개방 시간(ms)
Table 3. Node Opening Time

File system	depth-1	depth-2	depth-3
Linux	36	46	50
IOT-VFS	91	120	123

이를 보이고 있는데, 이 정도의 부담은 가상 파일시스템에 의해 얻어지는 다른 이점들에 의해 충분히 무시될 수 있는 수준이다. 특히, 개방 시간 자체는 데이터 입·출력 시간에 비하면 극히 미미하므로 극단적인 성능 저하가 아닌 한 가상 파일시스템의 장애요인이 될 수 없다.

Ⅴ. 결 론

IoT 디바이스는 인터넷이라는 접근 통로를 가진다는 점에서 공통적이지만, 그 기능이나 데이터의 형태, 데이터 접근 방법이 다양하다는 특징을 가지고 있다. 이 연구에서는 IoT 서비스 개발자나 데이터 사용자들이 IoT 디바이스의 다양성을 극복하고 POSIX 표준을 따르는 일관된 접근 인터페이스로 IoT 데이터에 접근할 수 있는 리눅스 FUSE 기반 IoT 플랫폼용 가상 파일시스템을 구현하고 평가하였다. 구현된 IoT 플랫폼에서는 데이터 입·출력이나 관리를 위한 별도의 API나 유틸리티 개발이 필요하지 않고, 성능 또한 충분히 수용 가능한 수준인 것으로 검증되었다. 개발된 IoT 플랫폼이 공개 소프트웨어 일환으로 더욱 발전하여 IoT 산업 부흥에 기여할 것으로 기대된다.

References

[1] M.-M. Kang, S.-R. Kim, "The Emerging Fusion Service," Communications of the Korean Institute of Information Scientists, Vol. 32 No. 2, pp. 9-21, 2014 (in Korean).

[2] J.-W. Shin, "Invitation to the IOT World," Digital News Article, Jul. 2014 (in Korean). http://www.denfoline.co.kr/news/articleView.ht

ml?idxno=8358
[3] H. Kim, D.-K. Kim, "Technology and Security of IoT," Journal of The Korea Institute of Information Security and Cryptology, Vol. 22, No. 1, pp. 7-13, 2012 (in Korean).
[4] Y.J. Park, M.H. Lim, G.J. Kim, "An analysis on the Market Trends and Demand of the RFID/USN Services," Electronics and Telecommunications Trends(ETRI), Vol. 24, No. 2, pp. 32-42, 2009 (in Korean).
[5] C.H. Liu, B. Yang, T. Liu, "Efficient naming, addressing and profile services in Internet-of-Things sensory environments," Ad Hoc Networks, Vol. 18, pp. 85-101, 2014.
[6] J. Kim, "IoT Platforms," Proceedings of 22nd Korea Internet Conference, 2014 (in Korean).
[7] S.T. Kim, J.S. Jeong, J.K. Song, H.Y. Kim, "Trends of IoT Device Platforms and Building its Ecosystems," Electronics and Telecommunications Trends(ETRI), Vol. 29, No. 4, pp. 82-90, 2014 (in Korean).
[8] P. Mell, T. Grance, "The NIST Definition of Cloud Computing," NIST, 2011, Online access: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
[9] J. Kim, J.S. Yun, S. Choi, M. Ruy, "Trends of IoT Platform Development and Evolution," Information Communications Magazine, Vol. 30, No. 8, pp. 29-39, 2013 (in Korean).
[10] Withings Blood Pressure Monitor, http://www.withings.com/en/bloodpressuremonitor
[11] Belkin WEMO, http://www.belkin.com/us/F7C029-Belkin/p/P-F7C029/
[12] GlowCap, http://www.glowcaps.com
[13] Arduino, http://www.arduino.cc
[14] MQTT, http://www.mqtt.org
[15] J. Kim, "IoT Platforms", Proceedings of The 22nd Korea Internet Conference, 2014 (in Korean).
[16] Xively, http://www.xively.com
[17] H.-B. Lee, K.-H. Kwon, "Implementation of a DB-Based Virtual File System for Lightweight IoT Clouds," KIPS Transactions on Computer and Communication Systems, Vol. 3, No. 10. 2014 (in Korean).
[18] IEEE Standard for Information Technology-Portable Operating System Interface, http://standards.ieee.org/develop/wg/POSIX.html
[19] FUSE, http://fuse.sourceforge.net

Hyung-Bong Lee (이 형 봉)

He received the B.S. and M.S. degrees in Computer Science from Seoul National University, Seoul, Korea, in 1984 and 1986 respectively. He received his Ph.D. degree in Computer Science from Kangwon National University, Chuncheon, Korea, in 2002. From 1986 to 1994, he was a senior engineer in Computer R&D Division of LG Electronics. From 1995 to 1998, he was with DEC(Digital Equipment Corporation) as a UNIX consultant. From 1999 to 2003, he was an Associate Professor at Honam University, Gwangju, Korea. Since 2004, he has been a Professor in the Department of Computer Science & Engineering at Gangneung-Wonju National University, Wonju, Korea. His current research interests include embedded systems, wireless sensor networks, and data mining algorithms.

Email: hblee@gwnu.ac.kr

Tae-Yun Chung (정 태 윤)

He received the B.S., M.S., and Ph.D. degrees in the School of Electrical & Computer Engineering at Yonsei University, Seoul, Korea in 1987, 1989, and 2000 respectively. From 1989 to 1996, he was a Research Engineer of Samsung Advanced Institute of Technology. From 1996 to 2001, he was with Samsung Electronics as a Senior Research Engineer. From 2000 to 2001, he was a vice chair of International DVD-Forum. Since 2001, he has been with the Department of Electronics Engineering at Gangneung-Wonju National University, Gangneung, Korea, and is currently a Professor. Since 2004, he has been with GEMS-CRC(Gangwon Embedded Software Cooperative Research Center, Gangneung, Korea) as the Chef. His major fields are image signal processing, digital video encoding, multimedia, copy protection , and his current research interests are embedded system, sensor network, video encoding.

Email: tychung@gwnu.ac.kr