Linux 2.6.x Kernel Device Model
(linux-2.6.13 기준)

Linux 2.6.x Kernel Device Model

2006년 01월 04일 고현철 정리

1. Introduction

커널 2.5 개발의 중요한 목표중 하나는 커널을 위한 통합 디바이스 모델이었다. 이전 커널에서는 시스템을 체계적으로 다룰 수 있는 자료구조가 존재하지 않았었다. 하지만, 점점 전원관리, hotplug등의 요구사항과 이에 대한 시스템 관리가 필요함에 따라 새로운 추상화 디바이스 모델이 필요로 하게 되었다.

"kobject" structure는 2.5.45 개발 커널에 처음선을 보였다. 초기에는 object에 대한 참조카운트를 관리하는데 사용되는 커널코드를 통합하는 단순한 기능을 가지고 있었다. kobject는 약간 별 것 아닌 기능을 가졌으나, 이제는 많은 디바이스 모델의 기능과 sysfs 인터페이스를 함께 통합하는 접착제와 같은 역할을 한다. driver 개발자가 kobject를 직접 다루는 일은 드문일이다: kobject들은 일반적으로 higher-level code에 의해 생성된 스트럭쳐 안에 숨겨져 있다.

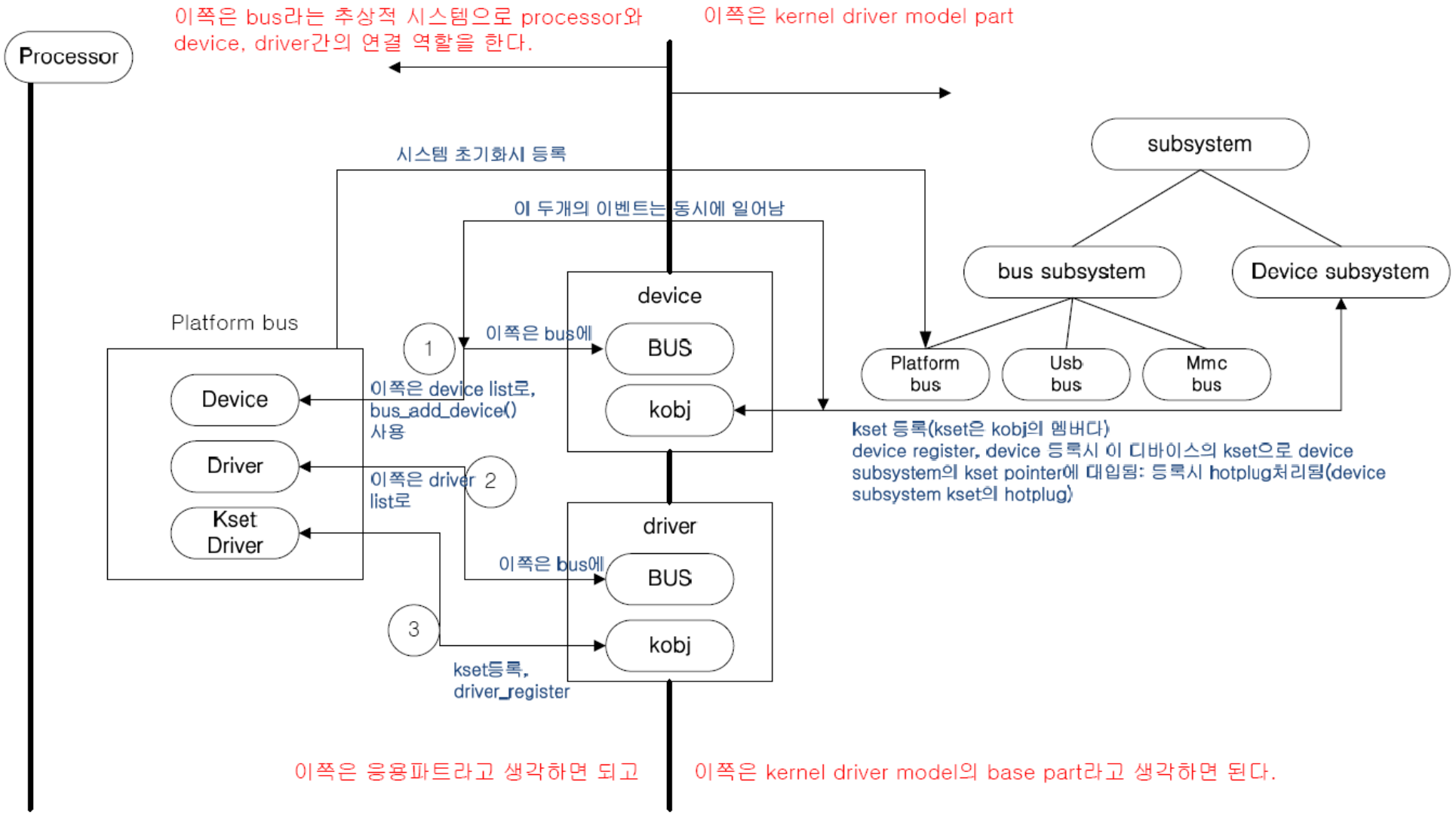
드라이버 모델을 이해하기 어려운 것 중 하나는(그리고 driver model을 만드는 kobject의 추상화에 대한) 명확한 시작점이 없다는데 있다(즉, 어디서부터 손을 대야할지 모르겠다는 말이다). kobject를 다루는 것은 서로 참조를 하고 있는 다른 type들에 대한 이해가 필요하다. 이해를 쉽게 하기 위하여 여러번에 걸친 접근을 한다. 애매한 단어로부터 시작을 해서 상세한 것을 덧 붙이는 형태로 접근한다. 여기서 우리가 다뤄야할 kernel driver model에 쓰이는 용어의 간단한 정의를 내리면 다음과 같다.

▶ kobject는 struct kobject 형의 object이다. kobject들은 이름과 참조카운트를 갖는다. kobject는 또한 parent 포인터(이 parent를 가지고 kobject는 계층구조로 배열이 된다)와 sysfs virtual filesystem에 나타나는 어떤 type(이게 ktype이군)을 갖는다. kobject는 일반적으로 그 자신에게 관심을 갖지 않는다: 대신 자기를 포함하는 실제로 사용이 되는 코드에 포함되는 어떤 다른 스트럭처에 관심을 갖는다.

▶ ktype은 kobject와 관계있는 type이다. ktype은 kobject가 더 이상 참조되지 않을경우 어떤일이 일어나야하는지와, kobject가 sysfs에서 어떻게 나타나야 하는지를 결정한다.

▶ kset은 같은 타입의 스트럭쳐에 내장된 kobject들의 그룹이다. kset은 kobject 집합의 기본적인 container type이다. kset은 그 자신도 kobject를 포함하고 있으며 다른 것들 사이에서(즉, kset에 붙어 있는 다른 kobject들) kobject들의 parent는 자기들이 포함되어 있는 kset의 kobject가 된다. 물론 이 kojbect들이 이런 방식을 갖지는 않지만....(쩝...아마도 kset에 있는 설명을 참조해야할 듯). sysfs의 모든 엔트리를 봤을 때 일반적으로, 그 엔트리들 각각은 같은 kset안에 있는 kobject들과 일치한다.

▶ subsystem은 서로 모여서 kernel의 중요한 sub-part를 구성하는 kset들의 집합이다. subsystem들은 보통 sysfs의 top-level 디렉토리와 일치한다.



1과 2: device와 driver의 등록은 등록순서와 관계없이 device 혹은 driver리스트를 서로 매칭되는지 검색해서 매칭되면 driver->probe()를 호출하고, device와 driver를 bind한다.

2와 3: 두 event(연결)는 bus_add_driver를 통해 이루어진다

2. The kobject Infrastructure

이 장은 kernel 2.6.13의 Documentation/kobject.txt를 번역하고, 약간의 설명과 코드를 수정 및 보완한 것이다(원래의 글과 실제 2.6.13 커널에서의 내용과는 다른 부분이 존재한다. 그 부분들을 되도록이면 2.6.13에 맞춰서 설명을 하였다. 하지만, 모든 것을 2.6.13으로 맞추지는 않고, 필요에 의해서만 그렇게 하였다. 자세한 코드등의 설명은 다음 장들을 참조하기 바란다.)

Patrick Mochel <mochel@osdl.org>

Updated: 3 June 2003

2.1. Introduction

kobject 구조는 기본적인 object 관리를 수행한다. 이 관리라는 것은 더 큰 데이터 구조와 서브시스템들을 비슷한 기능들을 재 사용하는 것보다 지렛대의 원리를 더 하듯이 더 많은 기능을 수행할 수 있도록 하는 것이다.

이 기능은 기본적으로 다음과 같은 것을 말한다.

- Object reference counting.
- Maintaining lists (sets) of objects.
- Object set locking.
- Userspace representation. (sysfs & udev관련)

내부구조는 이러한 기능을 지원하는 object의 타입들로 구성되어 있다. 이 object들을 프로그래밍하는 인터페이스들은 밑에 자세하게 설명하겠고, 간단하게 여기서 설명하면

- kobjects : 단순한 object를 말한다.
- kset : 어떤 특정 타입 오브젝트들의 set
- ktype : 일반적인 형태의 object에 대한 helper들의 set
- subsystem : 여러개의 kset을 다루는 object

kobject의 구조는 sysfs 파일 시스템과 밀접한 관계를 유지한다. kobject core에 등록된 각 kobject들은 sysfs 안에 디렉토리로 나타나게 된다. kobject에 관련된 특성들(attributes)은 밖으로 표시될 수 있다(sysfs를 통해서?). 자세한 내용은 Documentation/filesystems/sysfs.txt를 보면 된다.

kobject구조는 아주 유연한 프로그래밍 인터페이스를 제공하고, kobject와 kset들은 등록되지 않은 상태로 사용이 되는것도 허용한다(예를 들어 sysfs에 나타나지 않고 사용이 되는..), 이것 또한 뒤에 설명이 된다.

2.2. kobjects

2.2.1. Description

kobject 스트럭쳐는 더 복잡한 object 타입의 기초를 제공하는 단순한 데이터 타입이다. 이 스트럭쳐는 거의 모든 복잡한 데이터 타입에서 공유를 하고 있는 기본적인 필드를 제공한다(즉, 다른 모든 녀석들은 kobject란 필드를 가지고 있다는 얘기). kobject는 더 큰 데이터 스트럭쳐에 내장되는 경향이 있고, 그 필드들(kobject)이 duplicate되면서 필드를 다른 녀석으로 대체한다(kobject라는 필드를 가지고, 다른데 붙였다, 뗐다 한다는 말)

2.2.2. Defintion

```
struct kobject {
      const char         * k_name;
      char               name[KOBJ_NAME_LEN];
      struct kref        kref;
      struct list_head   entry;
      struct kobject     * parent;
      struct kset        * kset;
      struct kobj_type   * ktype;
      struct dentry      * dentry;
};

extern int kobject_set_name(struct kobject *, const char *, ...)
      __attribute__((format(printf,2,3)));

static inline const char * kobject_name(const struct kobject * kobj)
{
      return kobj->k_name;
}

extern void kobject_init(struct kobject *);
extern void kobject_cleanup(struct kobject *);

extern int kobject_add(struct kobject *);
extern void kobject_del(struct kobject *);

extern int kobject_rename(struct kobject *, const char *new_name);

extern int kobject_register(struct kobject *);
extern void kobject_unregister(struct kobject *);

extern struct kobject * kobject_get(struct kobject *);
extern void kobject_put(struct kobject *);

extern char * kobject_get_path(struct kobject *, int);
```

2.2.3. kobject Programming Interface

kobject는 kobject_register()와 kobject_unregister()를 이용해서 kobject core 에/로부터 동적으로 붙였다 제거했다 할 수 있다. 등록이라고 말하는 것은 이 kobject가 속해야 하는 kset의 리스트로의 추가와 sysfs의 디렉토리를 생성하는 것을 포함하는 말이다.

이와는 다르게, 사용자는 kobject_init()라는 함수를 단순하게 호출함으로써 kobject를 kset의 리스

트에 추가하지 않거나, sysfs를 통해서 보여지지 않도록 사용할 수 있다. 초기화된 kobject는 후에 kobject_add()를 호출해서 object 계층구조에 붙여 넣을 수 있다(즉, 2단계로 kset등에 포함시킬 수 있다는 얘기, 1번째는 생성, 두 번째는 add를 통해서). 초기화된 kobject는 참조 counting을 하는데 사용할 수 있다.

Note: kobject_init()를 호출하고 나서 kobject_add()를 호출하는 것은 kobject_register()를 호출하는 것과 같은기능을 한다.

kobject에 대한 등록을 해지할 때 이 kobject는 kset의 리스트와 sysfs 파일시스템으로부터 제거되고, 참조 카운트가 감소하게 된다. 리스트와 sysfs로부터의 제거는 kobject_del()에서 이루어지며, 수동으로 호출할 수 있다. kobject_put()은 참조 카운트를 감소시키며, 또한 수동으로 호출할 수 있다(즉, 자동호출이 아니라, 프로그래머가 임의적으로 호출할 수 있다는 얘기인 듯)

kobject의 참조 카운트는 kobject가 참조되고 있다는 것을 리턴해 주는 kobject_get()을 통해 증가시킬 수 있고, kobject_put()을 통해서 감소시킬 수 있다. object에 참조 카운트는 이미 positive일 경우만 증가된다.

kobject의 참조카운트가 0에 도달했을때 struct kobj_type::relaese()(kobject의 kset을 가리키고 있는 (kobj_type의))라는 메소드가 호출된다. 이것은 object에 대한 어떤 메모리 할당도 해제가 된다는 것을 의미한다.

NOTE!!!

만일 프로그래머가 kobject 참조 카운트를 사용했다면 release 한다는 것은 동적으로 생성된 kobject를 free하는데 사용되는 descructor를 반드시 제공해야 한다는 것이다. 참조 카운트는 오브젝트의 수명을 제어하는 것이다. 만일 0이 된다면 이 오브젝트는 해제가 될것이고, 사용되지 않을 것이라는 것을 말한다.

더 중요한 것으로, 프로그래머는 object를 unregister 호출을 한 바로 후가 아니라 반드시 바로 거기서(release()에서)만 해제해야 한다는 것이다. 만일 어떤 누가 object를 참조하고 있다면(예를 들어 sysfs file을 통해서), 참조한 녀석들은 object에 대한 참조를 획득하게 될 것이고, 그 오브젝트를 유효한 것으로 생각하고, 그것에 대해 작업을 해 버리게 될 것이다. 만일 그동안 오브젝트가 등록해제되고, free된다면, object에 대한 작업은 해제된 memory에 대한 참조가 될것이고 시스템은 “쾅”하고 날아가 버릴 것이다.

이러한 일은 그 참조 카운트가 0인 시점에서만 오브젝트에 대한 release method를 정의하고 free하게 함으로써 간단하게 막을 수 있다. 이러한 방법은 dual 참조 카운트를 사용하는 참조카운트/오브젝트 관리 모델처럼 안전하지는 않건, 참조카운트에 어떤 이상한 일을 하는 것 처럼 (networking layer에서처럼) 안전하지 않다는 것이다.(원문 볼 것)

2.2.4. sysfs

각 kobject는 sysfs안의 디렉토리를 할당받는다. 이 디렉토리는 kobject의 부모 디렉토리 밑에 생성된다.

만일 kobject가 등록되었을 때 부모를 갖지 않는다면 이 녀석의 부모는 이것이 속해있는 kset이 된다.

만일 kobject가 parent를 갖지 않거나, 이것이 속해 있는 kset도 없을 경우, 이 녀석에 대한 디렉토리는 sysfs 파티션의 top-level 디렉토리에 생성된다. 이것은 오직 struct subsystem안에 포함되어 있는 kobject들에 대해서만 일어나야 하는 일이다.(즉, sysfs의 top-leve 디렉토리에 나타나는 녀석들은 반드시 subsystem이어야 하는데, 이 subsystem 안에 있는 kobject란 녀석들은 부모도 없고, 속해잇는 kset도 없다는 얘기이다.)

2.3. kset

2.3.1. Description

kset은 같은 타입으로 내장된 kobject들의 집합니다.(같은 타입이라고 표현하지 않고, 같은 타입을 갖고 있는이라고 표현한 것은 그 kobject들이 kobject 자체로 존재하지 않고, 예를 들어 어떤 device등을 나타내는 것이기 때문인 것 같다. 해당 디바이스들은 kobject를 내장하고 있으니깐 ...^^)

가장 중요한 점을 설명하자면, 이 글 후에 설명이 되는 subsystem에 속해 있는 object의 경우 kset의 포인터를 가지고, 서로 링크를 만든다. 즉, subsystem의 kset의 ptr을 같은 object들의 kset이 가리키도록 만드는 것이다(ex> bus_register()의 경우 버스가 입력이 되면 bus subsystem의 kset의 ptr을 모든 입력된 bus의 kset으로 대입한다.)

그리고, kset의 경우 가장 중요한 것은 hotplug 처리가 kset에서 이루어진다는데 있다.

더 상세한 설명은 2장을 보면 된다. 여기(1장)는 원본의 틀을 최대한 유지하기 위하여 되도록이면 약간의 설명만하고 넘어간다.

코드는 다음과 같다.

```
struct kset {
       struct subsystem       * subsys;
       struct kobj_type       * ktype;
       struct list_head       list;
       spinlock_t             list_lock;
       struct kobject         kobj;
       struct kset_hotplug_ops * hotplug_ops; // ==> 이 부분은 2장서부터의 설명을 볼 것. 여기서는 pass.
..^^
};

void kset_init(struct kset * k);
int kset_add(struct kset * k);
int kset_register(struct kset * k);
void kset_unregister(struct kset * k);

struct kset * kset_get(struct kset * k);
void kset_put(struct kset * k);

struct kobject * kset_find_obj(struct kset *, char *);
```

kobject들이 내장되어 있는 타입은(즉, kobject가 속해 있는 디바이스등등의 타입?) ktype(kojb_type 형의) pointer에 기록되어 있다. 해당 kobject가 속해 있는 서브시스템은 subsys란 포인터에 의해 포인팅된다.

kset은 그 자체로 kobject를 포함하고 있는데, 그 의미는 kobject 계층구조에 등록이 되고, sysfs로 나타날 수 있다는 말이다. 더욱 중요하게, kset은 더 큰 데이터 타입에 내장될 수 있고, 다른 kset(그 object type(이말은 무슨말인지 알딸딸...--; ==> 같은 kobjec의 type을 가지는이란 말 같다. 밑의 설명을 볼 것))의 일부분으로 포함될 수도 있다.

예를 들어 block device는 block device의 set을 포함하는 object이다(struct gendisk라는 녀석은 block_device를 포함하는 녀석이란 얘기). 이것은 또한 device안에서 발견되는 partition(struct hd_struct)들의 set을 포함한다. 다음의 작은 코드는 어떻게 이런것들이 적절하게 표현되는지 보여준다.

```
struct gendisk * disk;
...
disk->kset.kobj.kset = &block_kset;
disk->kset.ktype = &partition_ktype;
kset_register(&disk->kset);
```

▶ disk의 내장 오브젝트가 속해 있는 kset은 block_set이고, 이 block_set은 disk->kset.kobj.kset을 가리킨다.(즉, disk의 멤버인 kset의 kobj의 kset은 block_set의 pointer란 얘기이다.)

▶ disk의 하부의 리스트에 있는 object의 type은 partitions이고, 이것은 disk->kset.ktype에 세팅된다.

▶ kset은 이제 등록이 되고, 계층구조에 내장된 kobject를 가지고 초기화되고 등록이 되도록 다뤄진다.

(뭐, kset이 kset에 세팅될 수 있다는 예제군....2.6.13 커널에서는 비슷한 코드 찾기 힘들다. genhd.c와 genhd.h를 봐도 여기서 설명된 것과는 틀린거 같다.)

2.3.2. kset Programming Interface

kset_find_obj()를 제외한 모든 kset 함수들은 kset에 관련된 동작을 한 후에 결국 내장하고 있는 kobject에 대한 호출로 이어진다. kset들은 kobject와 비슷한 프로그래밍 모델을 갖는다: kset들은 초기화 된 후에 kobject 계층구조에 등록하지 않은 상태에서 사용될 것이다(정확히는 이해하기 힘드나, 아마도 kobject_add()를 통해서 등록하기 전에 kset들에 대한 operation을 한다는 얘기인 것 같다. 물론 kset_add()의 경우 최종으로 kobject_add()를 호출한다 - lib/kobject.c)

kset_find_obj()는 특정 이름을 갖는 kobject의 위치를 찾는데 사용된다. 만일 발견된다면 kobject가 return된다.

2.3.3. kset관련 sysfs사항

kset들은 내장된 kobject들이 등록이 될 때 sysfs에 나타나게 된다. sysfs에 나타나는 kset들은 하나의 예외를 제외하고는 부모(parent)와 같은 룰을 따르게 된다. 만일 kset이 parent를 갖지 않거나, 내장된 kobject도 다른 kset의 일부분이 아닐 경우, kset의 부모는 kset 상위레벨로 kset을 지배하는(?....^^) subsystem이 된다(dominant subsystem)

만일 kset이 부모를 갖지 않는다면 kset에 대한 디렉토리는 sysfs root에 생성이 된다. 이것은 등록된 kset이 그 자체로 subsystem안에 내장되어 있을때만 허용이 되어야 한다. (즉, subsystem인 kset만이 가능하다는 얘기인 것 같다. 이건 코드를 좀 더 찾아봐야겠다.)

2.4. struct kobj_type(aka. ktype)

2.4.1. Description

```
struct kobj_type {
      void (*release)(struct kobject *);
      struct sysfs_ops    * sysfs_ops;
      struct attribute    ** default_attrs;
};
```

object tyes은 일반적인 object와 좀 더 복잡한 type간의 변환을 하는데 사용되는 특정 함수들을 요구한다. struct kobj_type은 다음과 같은 것들을 포함하는 object-specific 필드들을 제공한다.

- release: kobject의 참조 카운트가 0가 되었을때 호출된다. 이 함수는 object를 좀 더 복잡한 타입으로 변환시키고, 그 변환된 type형의 object를 free한다.

- sysfs_ops: sysfs 접근에 필요한 변환함수를 제공한다. 더 자세한 내용은 sysfs documentation을 보기 바란다.

- default_attrs: Default attributes는 object가 등록될 때 sysfs를 통해서 보여지게 된다. 마지막 특성(attribute)는 반드시 NULL로 초기화 되어야 한다는 것을 유념할 것. 이것에 대한 완벽한 구현은 drivers/block/genhd.c에서 찾아볼 수 있을것이다.

struct kobj_type가 실체로 나타나는 것(kobj_type의 instance는)은 등록되지 않는다. 오직 kset에 의해 참조만 될 뿐이다. kobj_type은 알 수 없는 만큼의 kset에 의해 참조될 수 있다. 동등한 object의 변종이 존재할 수 있기 때문이다(말 어렵다...--;).

2.5. subsystem

2.5.1. Description

subsystem은 다양한 타입을 갖는 object의 다양한 갯수의 set을 유지하는데 중요한 코드요소를 나타낸다. 많은 수의 kset과 많은 type의 object들이 다양한 변수들을 포함하고 있기 때문에 subsystem의 일반적인 표현은 최소가 된다(즉, kset과 kobj_type들이 실제로 다양한것들을 다 가지고 있고, 표현할 수 있으니, subsystem은 최소한만 갖고 있더도 된다는 얘기인 듯)

```
struct subsystem {
      struct kset        kset;
      struct rw_semaphore      rwsem;
};

int subsystem_register(struct subsystem *);
void subsystem_unregister(struct subsystem *);

struct subsystem * subsys_get(struct subsystem * s);
void subsys_put(struct subsystem * s);
```

subsystem은 내장된 kset을 가지고 있다. 그래서

- ▶ kset에 내장된 kobject를 통해서 object 계층구조에 나타날 수 있다.
- ▶ 하나의 타입을 갖는 object의 기본적인(default) list를 유지할 수 있다.(뭐, 한종류의 object list를 다룰 수 있다는 얘기인 듯)

부가적인 kset들은 kset이 등록되기 전엔 subsystem에 참조를 하게 함으로써 간단하게 subsystem에 붙을 수 있다.(이것은 일방적인 reference이고, 뜻하는 의미는 kset들이 subsystem에 attach되도록 결정할 방법이 없다는 것이다. - kset이 알아서 subsystem에 붙을 수 있는게 아니라 subsystem이 kset을 붙이는 거라는 말 같음)

subsystem에 붙어 있는 모든 kset들은 subsystem의 R/W semaphore를 공유한다.

2.5.2. subsystem Programming Interface.

서브시스템 프로그래밍 인터페이스는 단순하고, kset과 kobject 프로그래밍 인터페이스에서와 같은 유연성을 제공하지는 않는다. 참조카운트에서처럼 등록되고 등록해제를 할 뿐이다. 모든 호출은 subsystem이 내장하고 있는 kset에 대한 호출로 전달된다(그리고, 이것은 kset이 내장하고 있는 kobject에 전달 된다).

2.5.3. Helpers

몇 개의 매크로가 subsystem을 다루는데 유용하고, subsystem에 내장된 object들을 다루기 쉽게 만들어준다.

```
#define decl_subsys(_name,_type,_hotplug_ops) \
struct subsystem _name##_subsys = { \
      .kset = { \
            .kobj = { .name = __stringify(_name) }, \
            .ktype = _type, \
            .hotplug_ops =_hotplug_ops, \
      } \
}
```

이 decl_subsys매크로는 내장된 kset의 타입을 <type>으로 하고, '<name>_subsys'란 이름을 갖는 subsystem을 선언한다. 예를 들어 drivers/base/core.c의 devices subsystem은 다음과 같이 정의된다.

```
decl_subsys(devices, &ktype_device, &device_hotplug_ops);
```
이것은 다음과 같은 것이다:

```
struct subsystem devices_subsys = {
      .kset = {
           .kobj = {
                .name = "devices",
           },
           .ktype = &ktype_devices,
           .hotplug_ops = &device_hotplug_ops
      }
};
```

서브시스템에 등록이 되고, 서브시스템의 default list를 사용하는 object들은 적절하게 그들의 kset ptr을 반드시 설정해야 한다. 이것은 object들은 kobject, kset 혹은 다른 subsystem을 내장해야 하기 때문이다. 다음의 헬퍼는 kset 설정을 쉽게 해준다:

```
kobj_set_kset_s(obj,subsys)

- obj->kobj가 존재한다고 가정하고, 그것은 struct kobject 라야 한다.
- 그 kobject의 kset을 subsystem의 내장 kset으로 설정한다.

kset_set_kset_s(obj,subsys)

- obj->kset이 존재한다고 가정하고, 그것은 struct kset이다.
- 내장된 kobject의 kset을 서브시스템의 내장 kset으로 설정한다.

subsys_set_kset(obj,subsys)

- obj->subsys가 존재하고, 그것은 struct subsystem이라고 가정한다.
- obj->subsys.kset.kobj.kset을 서브시스템의 내장 kset으로 설정한다.
```

(실제 코드를 보면 obj에 대한 것을 subsys로 할당하는게 아니라, obj의 해당 부분(ex> kset)으로 subsys의 kset의 ptr을 할당하는 것이다. - kobject.h를 보면 된다.)

2.5.4. sysfs에서의 subsystem

subsystem들은 그들이 내장하고 있는 kobject들을 통하여 sysfs에 나타나게 된다. 서브시스템들은 어떤 예외도 없이 앞에서 언급한 같은 룰을 적용받는다. subsystem을 포함하는 kobject(혹은 subsystem에 포함된, 정확하게 어떤말일지는 약간 헷갈리지만 실제로는 같은 말이다)이 다른 kset의 일부분이거나, subsystem을 내장한 kobject의 부모가 명시적으로 설정이 되어 있을때를 제외하곤 일반적으로 sysfs의 top-level 디렉토리를 배정받는다.

Note: subsystem의 내장 kset은 서브시스템의 rwsem을 이용하기 위해서는 반드시 subsystem 자체에 붙어 있어야 한다는 것을 명심할 것. 이것은 kset_add()를 호출한 후에 이루어진다(호출전이 아니다. 왜냐하면 kset_add()는 이미 부모를 갖지 않고 있을 경우 default parent로 그 자신의 subsystem을 사용하기 때문이다.)

// kobject.c를 보면 다음과 같이 되어 잇다.

```
int kset_add(struct kset * k)
{
       if (!k->kobj.parent && !k->kobj.kset && k->subsys)
              k->kobj.parent = &k->subsys->kset.kobj;
```

```
        return kobject_add(&k->kobj);
}
```

3. kobject.h code description

3.1. Embedding kobjects

```
- include/linux/kobject.h

#define KOBJ_NAME_LEN   20

/* counter to tag the hotplug event, read only except for the kobject core */
extern u64 hotplug_seqnum;

struct kobject {
        const char            * k_name;
        char                  name[KOBJ_NAME_LEN];
        struct kref           kref;
        struct list_head      entry;
        // 이 kobject의 상위층에 존재하는 object를 나타냄
        // 만일 이 kobject가 usb hub에 붙은 device장치(ex> mass storage)를 나타낸다면
        // parent는 usb hub를 얘기하는 것이 된다.
        struct kobject        * parent;
        struct kset           * kset;     // 이 object가 속해 있는 kset
        struct kobj_type      * ktype;
        struct dentry         * dentry;
};

kobject를 release하는 함수가 필요하게 되는데(참조가 0일때 동작해야하는) 이 함수는
비동기적으로 동작해야 한다. 왜냐하면 드라이버가 언제 없어질지 모르기 때문이다.

일반적으로 알고 있지만, usb나 mmc같은 넘들은?.....그냥 뽑아버릴 수도 있기 땜시...^^
(아마도 이말 같다....)

이 release 함수는 kobject에는 없지롱!


// kobj_type에 대한 설명은 뒤에 자세하게 한다. 대충 읽고, 뒷 부분을 참조할 것.

어디 있냐하면 ktype으로 불리우는 struct kobj_type에 존재한다.

struct kobj_type {
        void (*release)(struct kobject *);
        struct sysfs_ops        * sysfs_ops;
        struct attribute        ** default_attrs;
};

kobject structure를 보면 ktype을 참조하는데가 두군데이다. 하나는 안 보이는데
그녀석은 이 밑의 kset을 보듯이 kset의 멤버인 ktype이 또 있다.

만일 kobject가 kset에 포함되어 있지 않다면 kobject자체내의 ktype을 참조하고,
kset에 포함되어 있다고 한다면 kset의 ktype을 참조하게 된다.
```

kernel코드에서 stanealone상태의 kobject를 생성하는 것은 드문일이다; 대신 kobject는 좀 더 크고, 자체적으로 특별한 object에 접근제어하는데 사용된다. 이 끝에 kobject는 다른 스트럭쳐에 내장되어 있는 것을 발견할 수 있을것이다. object-oriented의 입장에서 생각을 해보면, kobject들은 다른 클래스를 파생시키는 최상위의 추상적인 클래스라고 볼 수 있다. kobject는 그 자체로는 특별하게 유용하지 않도록 설계된 것이지만, 다른 오브젝트들에 포함이 되면 굉장히 유용하다. C 언어가 상속에 대한 직접적인 표현을 허용하지 않기 때문에 다른 테크닉-스트럭쳐 내장형과 같은-이 사용되

어야 한다.

Hooking into sysfs

초기화된 kobject는 문제없이 참조카운트를 다루게 되지만, sysfs에 나타나지는 않는다. sysfs 엔트리를 만들기 위해서, kernel 코드는 반드시 해당 object를 가지고 kobject_add()를 호출해야 한다:

```
int kobject_add(struct kobject *kobj);
```

항상 그렇듯이 이 함수는 실패할 수 있다.

```
void kobject_del(struct kobject *kobj);
```

이 함수는 sysfs로부터 kobject를 제거한다.

kobject_init()와 kobject_add()를 합쳐놓은 kobject_register()가 있고, 비슷하게 kobject_unregister()는 kobject_del()과 kobject_register()(혹은 kobject_init())에서 생성된 reference를 release하는데 사용되는 kobject_put()을 호출할 것이다.

3.2. container_of macro

kobject를 가지고 여러 가지 device등을 다룰 경우 이 매크로가 많이 쓰인다. 여기서는 kobject의 용법으로 사용되지는 않았지만, kobjetct를 struct device로 변경할 경우 많이 사용되는 매크로이다.

예를 들어

```
- drivers/base/core.c
#define to_dev(obj) container_of(obj, struct device, kobj)
```

같은 것이다.

```
- kernel.h
/**
 * container_of - cast a member of a structure out to the containing structure
 *
 * @ptr:        the pointer to the member.
 * @type:       the type of the container struct this is embedded in.
 * @member:     the name of the member within the struct.
 *
 */
#define container_of(ptr, type, member) ({                  \
        const typeof( ((type *)0)->member ) *__mptr = (ptr);    \
        (type *)( (char *)__mptr - offsetof(type,member) );})
```

ptr : 멤버의 포인터

type: 이 멤버를 포함하고 있는 컨테이너 스트럭쳐의 타입

member: type structure에 안에서 존재하는 멤버의 이름

ex> s3c2410 framebuffer의 경우

```
struct s3c2410fb_info {
```

```
    struct fb_info        fb;
    struct device         *dev;
    struct clk            *clk;

    struct s3c2410fb_mach_info *mach_info;

    /* raw memory addresses */
    dma_addr_t            map_dma;     /* physical */
    u_char *        map_cpu;     /* virtual */
    u_int                 map_size;

    struct s3c2410fb_hw   regs;

    /* addresses of pieces placed in raw buffer */
    u_char *        screen_cpu;  /* virtual address of buffer */
    dma_addr_t            screen_dma;  /* physical address of buffer */
    unsigned int          palette_ready;

    /* keep these registers in case we need to re-write palette */
    u32                   palette_buffer[256];
    u32                   pseudo_pal[16];
};
```

라는 s3c2410 고유의 스트럭쳐가 있을 경우 framebuffer main frame에서는 실제로

```
struct fb_info           fb;
```

만을 사용한다. 하지만 드라이버에서는 s3c2410 고유의 스트럭쳐를 필요로 한다.

해서 fb만을 가지고 s3c2410fb를 가져와야 한다.(실제로는 fb의 ptr을 가지고 해야한다. 왜냐하면 등록할때 s3c2410fb의 fb의 ptr을 가지고 등록하기 때문이다.)

해서 다음과 같은 inline함수를 만들어서 사용한다.

```
static inline struct s3c2410fb_info *fb_to_s3cfb(struct fb_info *info)
{
    return container_of(info, struct s3c2410fb_info, fb);
}
```

입력은 당연히 fb_info 형 ptr이 되고 출력은 이 녀석을 품고 있는 s3c2410fb_info structure의 ptr을 리턴해 줘야 한다.

맨 마지막 인자인 fb는 s3c2410fb_info에서의 fb_info라는 형을 가지는 녀석의 이름이 fb라고 얘기하는 것이다.

3.3. ktype과 release methods

위의 논의에서 여전히 빠져있는것들 중 중요한 것은, kobject의 참조 카운트가 0이 되었을 때 kobject에 어떤일이 일어나는가 하는 것이다 kobject를 생성한 코드는 일반적으로 언제 이런 일이 일어날지를 모른다; 만일 그렇다 하더라도, 맨 처음 시점에서의 kobject의 사용에는 별 도움이 안된다. object의 생명의 예측이 가능하더라도 sysfs에서 그것을 없애고자 했을 때 매우 복잡하게 된다: 어느 기간동안 사용자 영역 프로그램이 여전히 kobject를 참조하고 있을경우(kobject와 관련있는 sysfs에 있는 file을 open을 하고 유지할 경우)

최종결론으로, (kobject) structure는 그 참조 카운트가 0으로 되기전에 free 될 수 없도록 보호 되어야 한다는 것이다. 참조카운트는 kobject를 생성하는 코드의 직접적인 영향하에 있지 않다. 그래서 해서 해제코드는 반드시 kobject에 대한 마지막 참조가 없어진 어느 시점에서나 비 동기적으로 인지를 받아야 한다.

이러한 인지방법은 kobject의 release() method를 통해서 이루어진다. 일반적으로 이러한 method는 다음과 같은 형태를 갖는다:

```
void my_object_release(struct kobject *kobj)
{
    struct my_object *mine = container_of(kobj, struct my_object, kobj);

    /* Perform any additional cleanup on this object, then... */
    kfree (mine);
}
```

한가지 중요한 점은 미리 앞서서 처리하지 말아야 한다는 것이다(남용하지 말란 얘기?): 모든 kobject는 반드시 release() method를 가져야 하고, kobject는 반드시 method가 호출될 때까지는 유지되어야 한다. 만일 이러한 제약이 없을 경우, 코드는 불안하게 되어 버릴 것이다.

재밌게도 release() method는 kobject 자체내에 존재하지 않는 것이다; 대신 ktype에 포함되어 있다:

```
struct kobj_type {
    void (*release)(struct kobject *);
    struct sysfs_ops  *sysfs_ops;
    struct attribute  **default_attrs;
};
```

이 스트럭쳐는 kobject의 특정한 type(혹은, 더 정확히 object를 포함하는 type)을 표현하는데 사용된다. 모든 kobject는 kobj_type을 가져야할 필요가 있다; 이 스트럭쳐(ktype)에 대한 포인터는 초기화할 때 kobject의 ktype 필드에 들어있거나 (좀 더 가깝게 얘기하면)kobject를 포함하고 있는 kset에 의해 정의될 수 있다.

kobj_type의 release 필드는 물론 이 kobject type의 release() method이다. 다른 두개의 필드(sysfs_ops & default_attrs)는 어떻게 이 타입의 object가 sysfs에 보여져야 하는지를 결정한다; 이 부분은 sysfs에 대한 문서를 참조해야 한다.

3.4. kset

```
- include/linux/kobject.h
/**
 *     kset - a set of kobjects of a specific type, belonging
 *     to a specific subsystem.

 특정 subsystem에 포함되어 있는 특정 타입의 kobject의 set

 *
 *     All kobjects of a kset should be embedded in an identical
 *     type. This type may have a descriptor, which the kset points
 *     to. This allows there to exist sets of objects of the same
 *     type in different subsystems.
```

```
kset에 있는 모든 kobject들은 동등한 type으로 포함되어 있어야 한다.
이 타입은 kset이 가르키고 있는 디스크립터를 가져야 한다.

이것은 다른 서브시스템에 있는 같은 타입의 오브젝트들의 set들이 존재하도록
해준다.

(다른 서브시스템에 붙어 있어도 같은 타입의 것들이 모여 있는 kset들을
묶을 수 있다는 말인가?)

 *
 *     A subsystem does not have to be a list of only one type
 *     of object; multiple ksets can belong to one subsystem. All
 *     ksets of a subsystem share the subsystem's lock.
 *

서브 시스템은 오직 한 타입의 object들의 리스트를 가져서는 안된다: 이것은
여러개의 kset이 하나의 subsystem에 속해 있다는 말이다. 서브시스템의 모든
kset들은 subsystem의 lock을 공유한다.

 *     Each kset can support hotplugging; if it does, it will be given
 *     the opportunity to filter out specific kobjects from being
 *     reported, as well as to add its own "data" elements to the
 *     environment being passed to the hotplug helper.

각 kset들은 hotplugging을 지원한다; 만일 지원한다면, 보고된(hotplug상황이?)
것으로부터 특정 kobject들을 filtering할 수 있는 기회가 주어진다.
이와 마찬가지로 hotplug helper로 object 자신만의 "data" 구조를 환경변수로
(environment로) 전달할 수 있는 기회도 갖는다.

==> 음, filtering도 가능하고, 몇가지 더 덧붙이는것도 가능하다는 말 같군....에라...

 */
struct kset_hotplug_ops {
       int (*filter)(struct kset *kset, struct kobject *kobj);
       const char *(*name)(struct kset *kset, struct kobject *kobj);
       int (*hotplug)(struct kset *kset, struct kobject *kobj, char **envp,
                      int num_envp, char *buffer, int buffer_size);
};

struct kset {
       struct subsystem       * subsys;
       struct kobj_type       * ktype;
       struct list_head       list;
       spinlock_t             list_lock;
       struct kobject         kobj;
       struct kset_hotplug_ops * hotplug_ops;
};
```

많은 경우 kset은 kobj_type 스트럭쳐의 확장처럼 보인다: 원래 kset은 같은기능의(identical인데, 동등한 혹은 같은성격을 갖는) kobject들의 모임이다. 하지만, strucut kobj_type은 그 자체로 object type에만 관련있는 반면에 struct kset은 집합과 모임에 관련된다. 같은 타입의 이 두개념은 같은 type의 object라도 다른 set에 나타날 수 있기 때문에 분리되어야 한다(ktype은 내용적인 개념에 대한 얘기이고, kset은 등록에 대한 얘기이다. 예를 들어 같은 type의 device도 다른 bus(의 kset)에 속해 있을 경우가 있기 때문이다. 이 경우 type은 같아도, 실제로 소속이 틀리게 된다:다른 kset에 소속).

kset은 다음과 같은 기능을 제공한다:

▶ 동일한(같은종류의) object들의 그룹을 포함하는 가방과 같은 역할을 한다. kset은 커널에서 “모든 block devices" 혹은 ” 모든 PCI device drivers"와 같이 사용될 수 있다.

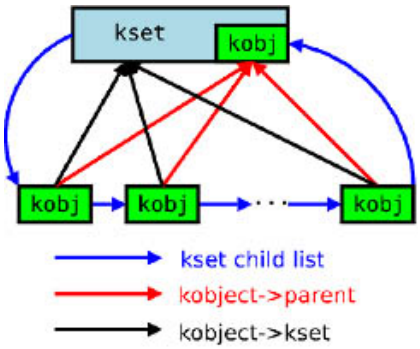
▶ kset은 device model(과 sysfs)을 함께 간직하도록 하는 디렉토리 레벨의 접착제와 같은 것이다(즉, kset을 기점으로 해서 모든 동질의 kboject들이 tree형태로 모이게 한다는 말인것 같다- 이것은 sysfs의 디렉토리에서도 마찬가지로 나타난다) 모든 kset은 다른 kobject들의 parent로 설정될 수 있는 kobject를 포함한다: 이런 방법으로 device model hierarchy가 만들어진다.

▶ kset은 kobject의 "hotplugging"을 지원하고, 어떻게 event가 사용자 영역으로 전달되어야 하는지에 대해 영향을 준다(ex> filtering & 환경변수 추가 등등....)

객체 지향적인 측면에서 "kset"은 class의 top-level container이다; kset은 kset 자체의 kobject로부터 상속된 넘이고, kset 그 자체도 kobject로 취급될 수 있다.

kset은 자식들을 전형적인 kernel linked list로 유지한다. kobject들은 그 자체의 kset 필드를 가지고 자기 자신을 포함하는 kset을 역으로 가리킬 수 있다. 또 대부분의 경우 이 kset에 포함된 kobject들은 그들의 parent filed에서 kset(혹은 엄밀히 kset이 내장하고 있는 kobject)을 가리키고 있게 된다(즉, kset에 포함된 kobject->parent->kset은 kobject->kset이란 얘기이다).

그래서, 일반적으로 kset과 kset의 kobject들은 밑의 그림에서 처럼 보인다.



kset-sm.png

다음을 유념하기 바란다.

(1) 그림에서 kset에 포함된 모든 것들은 실제적으로는 이 kobject를 내장하는 다른 타입의(다른 kset조차 가능)것이다.

(2) 여기서 등록된 kobject들의 parent가 kset을 포함할 필요는 없다.

kset을 초기화하고 설정하는데 필요한 인터페이스는 kobject와 매우 유사하다.

다음과 같은 함수가 존재한다:

```
void kset_init(struct kset *kset);
int kset_add(struct kset *kset);
int kset_register(struct kset *kset);
void kset_unregister(struct kset *kset);
```

이 함수들은 대부분 kset에 포함되어 있는 kobject에 대한 kobject_ 함수와 유사하게 호출된다.

kset의 참조 카운트를 다루는 상황에서도 비슷하다:

```
struct kset *kset_get(struct kset *kset);
void kset_put(struct kset *kset);
```

kset도 역시 내장된 kobject에 이름을 갖는다. 만일 my_set이라는 kset은 다음과 같이 이름을 설정할 수 있다:

```
kobject_set_name(my_set->kobj, "The name");
```

또한 kset은 자기가 포함하고 있는 kobject들에 대한 특성을 표현해주는 kobj_type 스트럭쳐에 대한 포인터를 갖는다(ktype 필드). 이타입은 자체적으로 kobje_type structure를 갖고 있지 않는 모든 kobject에 적용이 된다.

kset의 다른 특징 중 하나는 hotplug operation이다; 이 operation은 kobject가 kset으로 들어가거나/나올 때 항상 호출된다(실제 코드에서는 device_register or driver_register /device_unregister or driver_unregister 시). 이 hotplug 기능은 user-space hotplug event를 발생할 것인지, 어떻게 그 이벤트가 영향을 미칠지를 결정한다(어떤식으로...). hotplug에 대한 자세한 내용은 sysfs쪽에서 다루어진다(lwn.net의 driver-porting부분의 sysfs에서) - 뭐 여기서는 다른부분의 코드에서 다루면 된다.

어떤 사람들은 다음과 같은 질문을 한다. kobject를 kset에 붙이는 기능을 하는 함수가 없다고, 그런 기능을 하는 함수가 주어지지 않았다고. 답으로는 "그 일은 kobject_add()가 담당한다"이 된다. kobject가 kobject_add()로 전달되었을 때, kobject의 kset 멤버는 kobject가 소속되어야 할 kset을 가리키고 있다. kobject_add()는 그 나머지를 다룰 것이다(즉, 먼저 세팅된 채로 들어와야 한다는 말이다, driver_register()를 추적해보면 간단하게 알 수 있다). 현재로서 리스트 포인터들을 헝클어 뜨리지 않고, 직접적으로 kset에 kobject를 붙이는 어떠한 방법도 존재하지 않는다.

마지막으로 kset은 subsystem pointer(subsys라고 불리는)를 갖고 있다.

3.5. subsystem

```
struct subsystem {
       struct kset           kset;
       struct rw_semaphore    rwsem;
};


// subsystem 선언 ==> 만일 devices_subsys를 선언하고 싶다면
// decl_subsys(devices, &ktype_device, &device_hotplug_ops);
// 이렇게 하면 된다.(관련 자료는 분석자료_mmc_sysfs_hotplug관련.txt를 보면된다.)
// 즉 subsystem으로 선언되어 있는 devices_subsys라는 변수를 아무리 찾아도 안나오는
// 이유는 macro를 써서 만들어 버리기 때문이다.
#define decl_subsys(_name,_type,_hotplug_ops) \
struct subsystem _name##_subsys = { \
       .kset = { \
              .kobj = { .name = __stringify(_name) }, \
              .ktype = _type, \
              .hotplug_ops =_hotplug_ops, \
       } \
}
#define decl_subsys_name(_varname,_name,_type,_hotplug_ops) \
```

```
struct subsystem _varname##_subsys = { \
	.kset = { \
		.kobj = { .name = __stringify(_name) }, \
		.ktype = _type, \
		.hotplug_ops =_hotplug_ops, \
	} \
}
```

subsystem은 (그 전체로서) 커널의 high-level 부분을 나타낸다.

위에서 볼 수 있듯이 subsystem은 간단한 스트럭쳐를 갖는다.

subsystem은 위의 구조에서 보듯이 실제로는 그냥 kset을 wrapping하는 녀석이다. 사실 subsystem의 기능은 단순하다: 하나의 subsystem은 여러개의 kset을 담을 수 있다. 이러한 것은 kset 스트럭쳐의 subsys pointer에 의해서 이루어진다; 그래서, 만일 subsystem에 여러개의 kset이 있다면, 서브시스템 구조에서 그 kset들 모두를 직접적으로 찾아낼 방법이 없다.(이건 코드를 봐야 알 수 있는 부분인데, subsystem이 kset을 접근하는 방식이 아니라, 일방적으로 kset->subsys = &(subsystem.kset), 이런 방식으로 대입을 해 넣는다. 그러니깐, subsystem을 가지고서는 거기에 붙어 있는 kset들을 찾아내지 못한다.)

Every kset must belong to a subsystem; the subsystem's rwsem semaphore is used to serialize access to a kset's internal linked list.

모든 kset은 반드시 subsystem에 속해야한다; 서브시스템의 rwsem semaphore는 kset의 내부 linked list에 serialize하게(원래는 번역해야하나, 이 말이 가장나은거 같아서 그냥 냅둔다...쩝) 접근하는데 사용된다.

서브시스템은 다음과 같은 macro를 가지고 종종 선언된다(내가 보기엔 100%인데...ㅋㅋ):

```
decl_subsys(char *name, struct kobj_type *type, struct kset_hotplug_ops *hotplug_ops);
```

이 매크로의 사용법은 코드를 직접보거나, 코드설명해 놓은 부분을 보면 되겠다.

서브시스템들은 setup을 하거나, 해제하는데 다음과 같은 함수를 사용한다:

```
void subsystem_init(struct subsystem *subsys);
int subsystem_register(struct subsystem *subsys);
void subsystem_unregister(struct subsystem *subsys);
struct subsystem *subsys_get(struct subsystem *subsys)
void subsys_put(struct subsystem *subsys);
```

이 함수들의 대부분의 동작은 subsystem의 kset에 대해 이루어진다.

4. Examining a kobject hierarchy

- [Posted October 29, 2003 by corbet]
- 원문: http://lwn.net/Articles/55847/

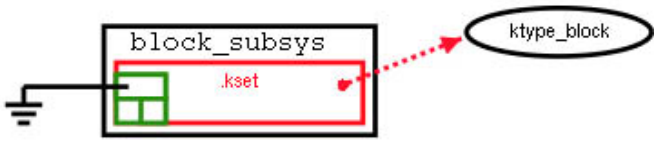
이 장의 내용은 lwn.net에 올라와 있던 위의 기사를 번역하면서 약간의 수정과 편집을 하였다. 전체 시스템에 대한 설명은 아니고, block subsystem에서 일어나는 kobject hierarchy의 구성을 일례로 든 것이다.

LWN.net의 Driver Porting Series는 어떻게 kobject가 서로의 데이터 스트럭처를 묶고, 참조 카운트를 다루는지 여러 기사에 걸쳐 다뤘다. 여러 예를 들어서 설명했으나, 실제로 kobject가 서로 얽혀있는 데이터 구조를 실제로 상상하거나 그리는 일은 굉장히 어려운 일이다. 이 기사는 block layer에서 data structure가 어떻게 엮여서 서로 구성되어 있는지를 보여준다.

핵심적인 데이터 구조는 kobject이다. 다음에 오는 다이어 그램에서 kobject는 다음과 같은 자그마한 심볼로 표현된다. 윗부분의 사각형은 kobject의 parent 필드를 가르키고, 다른 두개는 kset에 구현되어 있는 양방향 링크드 리스트 요소를 나타낸다. 여기서 조심할 점은 모든 kobject들이 kset에 속한게 아니기 때문에, kset에 속하지 않는 넘들은 가끔 빈상태로 되어 있다.

kobject.png

block subsystem hierarchy의 최상위 부분(천정)은 subsystem인 block_subsys이다; 이 녀석은 drivers/block/genhd.c에 선언되어 있다. (The Zen of Kobjects기사를 보면) subsystem은 세마포어와 kset으로 이루어진 매우 단순한 스트럭쳐이다. kset에는 ktype이란 필드안에 kobject들의 타입이 저장되게 되어 있다(즉, 이 kset은 ktype을 갖는 kobject들의 모임이다). 여기서 예를드는 block_subsys의 경우 이 ktype 필드는 ktype_block으로 설정된다. 그림으로 봤을 때 오른쪽에 ktype_block으로 표시된 녀석이다.



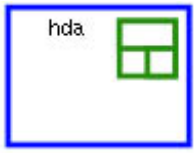
block-subsys.png

각 kset들은 자체로 kobject를 포함하고 있고, block_subsys도 예외는 아니다. 이 경우 kobject의 parent 필드는 명시적으로 NULL이 되어야 한다(그림에서는 ground로 표시되어 있다...^^). 이렇게 설정함으로써 이 kobject는 sysfs 구조에서 top-level에 나타나게 된다: 즉, 이 kobject는 /sys/block 뒤에 가만히 있는 녀석이 된다(그냥 존재한다는 얘기인 듯)

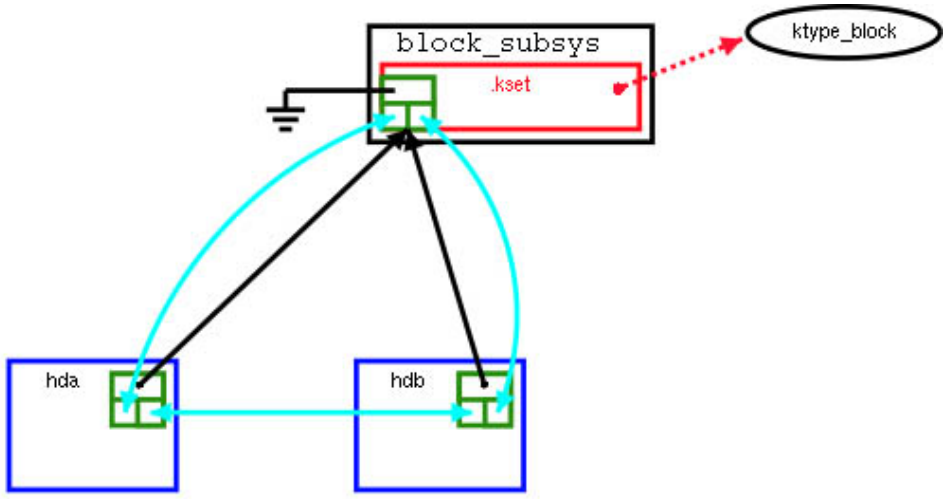
But its parent and kset pointers both point to the kobject within block_subsys, and the kset pointers are there too. The result, for a system with two disks, would be a structure that looks like this:
disk 없는 block subsystem은 별 의미가 없다. block 계층구조(hierarchy)에서, 디스크는 struct gendisk로 표현된다(include/linux/genhd.h). gendisk 인터페이스는 http://lwn.net/Articles/25711 기사에서 볼 수 있다. 여기서는 그림을 그리는게 목적이기 때문에 gendisk를 밑의 그림과 같이 나타내었다(disk-box.png). 여기서 주의해야 할 점은 반드시 kobject를 가지고 있어야 한다는 것이다.

gendisk의 kobject는 명시적으로 pointer형이 아니다; block_subsys kset이 그것을 다루게 된다(이것을 코드로 봐야할겠지만, 느낌상으로는 kset의 kobj pointer로 gendisk안의 멤버인 struct kobject kobj로 대입이 되지 않나 생각이 든다). 그러나, 그 kobject의 (parent와 kset pointer) 둘다 block_subsys 안의 kobject를 가리키고, kset pointer도 마찬가지로 거기를 가리킨다(이 말은 좀 이상하다. drivers/block/genhd.c의 add_disk()를 추적해서 , fs/partitions/check.c의 register_disk()로 점프를 하면 그냥 gendisk의 kobj를 가지고 kobject_add()를 한다. 거기서 보면 parent와 kset 세팅만 할 뿐이지, 실제 gendisk에 kset ptr이 없기 때문에 그냥 kobject의 연결로만 마무리가 된다. 시간이 지난기사라 코드가 그동안 바뀌었을지 모르겠다. 2.6.13에 그런 코드는 없는 듯 해 보이지만, 하여간 그림상으로는 맞는거 같다). 결과적으로 두개의 디스크가 있는 시스템에서, 그 구조는 다음과 같이 된다(disks-only.png). (여기서 cyan색으로된 양방향 링크드 리스트를 보면 gendisk의 kobject의 parent로 block_subsys의 kset의 kobject가 된 것을 알 수 있고, 서로는 linked list로 되어 있는 것을 볼 수 있다. 이것은 코드상으로 위에서 설명한 register_disk()에서 kobject_add()로 흘러가는 코드를 보면 알 수 있다.)

```
// include/linux/genhd.h
struct gendisk {
	int major;			/* major number of driver */
	int first_minor;
	int minors;                     /* maximum number of minors, =1 for
                                         * disks that can't be partitioned. */
	char disk_name[32];		/* name of major driver */
	struct hd_struct **part;	/* [indexed by minor] */
	struct block_device_operations *fops;
	struct request_queue *queue;
	void *private_data;
	sector_t capacity;

	int flags;
	char devfs_name[64];		/* devfs crap */
	int number;                     /* more of the same */
	struct device *driverfs_dev;
	struct kobject kobj;

	struct timer_rand_state *random;
	int policy;

	atomic_t sync_io;		/* RAID */
	unsigned long stamp, stamp_idle;
	int in_flight;
#ifdef  CONFIG_SMP
	struct disk_stats *dkstats;
#else
	struct disk_stats dkstats;
#endif
};
```

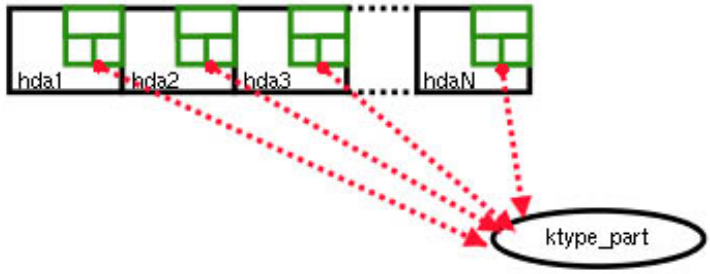


disk-box.png



disks-only.png

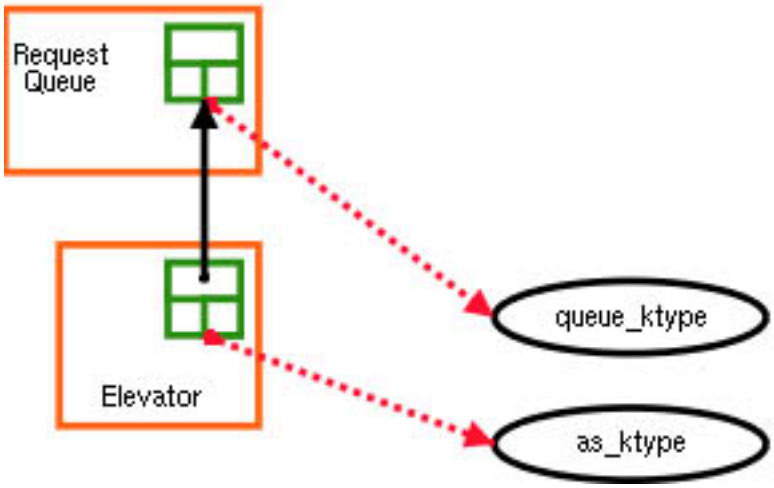
여기서 끝은 아니다. 왜냐하면 gendisk 구조는 복잡한 넘이기 때문이다. gendisk는 partition이라는 요소를 배열로 포함할 수가 있는데(struct hd_struct), 이 스트럭쳐 또한 그 안에 kobject를 포함하고 있다. 각 파티션들의 parent는 자기를 포함하는 disk가 된다. 여기서 partiton들도 kset의 리스트로 구현할 수 있다. 하지만 그렇게 구현하지는 않았다. partition들은 상대적으로 static한 item들이고, 순서가 매겨져 있는 넘들이기 때문에 그냥 간단한 array로 구현을 하였다. 그 array는 다음의 그림에서 볼 수 있다.



partitions.png

그림에서 볼 수 있듯이 partition의 kobject type은 ktype_part이다. 이 타입은 각 파티션에 대해 시작되는 block number와 크기를 포함하는 각 정보를 sysfs에 볼 수 있게 해주는 attribute를 가지고 있다.

각 gendisk에 함께 연결된 다른 아이템은 해당 gendisk의 I/O request queue이다. 이 큐도 역시 그 parent는 gendisk인 kobject(object의 type은 queue_ktype이다)를 포함한다. 그림에서 발견할 수 있는 또 다른 하나는 I/O request queue와 함께 사용되는 I/O 스케줄러(“elevator"라고 불리는?)이다. 스케줄러의 kobject 타입은 스케줄러가 사용되는 용도에 따라 다르다: (기본적으로 사용되는) anticipatory scheduler는 as_ktype이다. 지금 설명되는 퍼즐의 결과는 다음과 같이 그릴 수 있다.



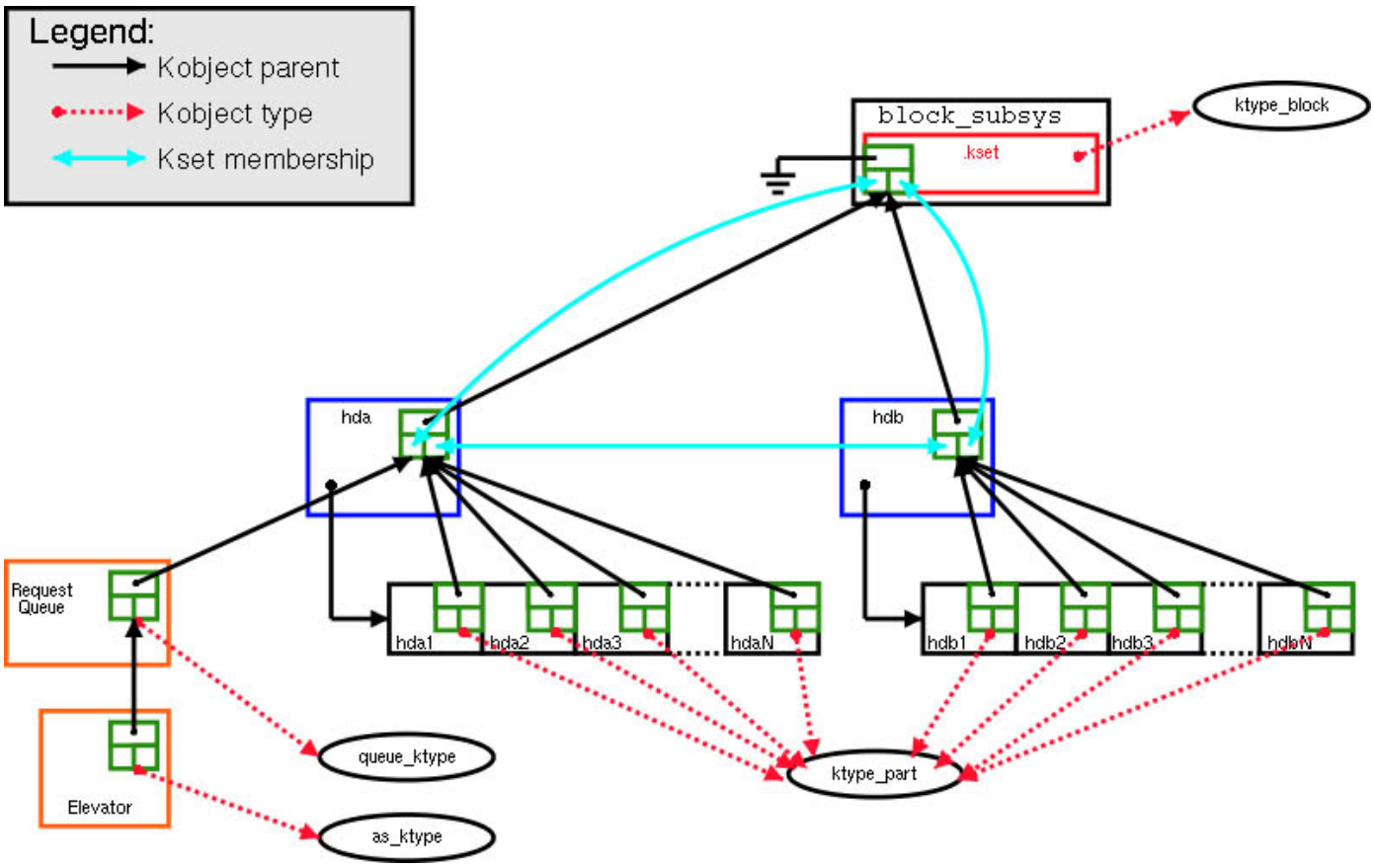
iorq.png

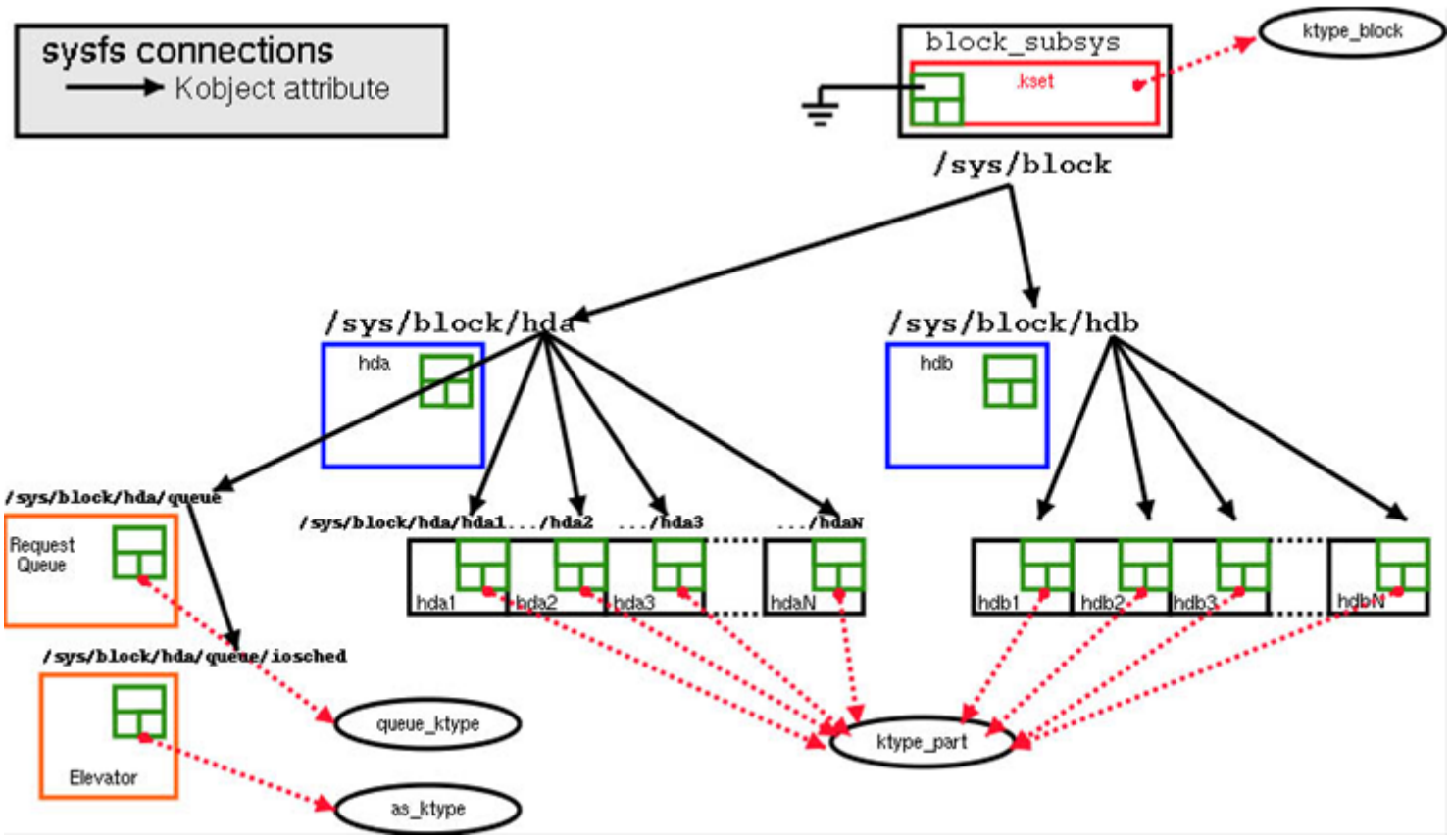
sysfs에서 request queue와 I/O 스케줄러 정보는 현재 read-only이다. 이유는 없지만 왜 sysfs의 특성(attributes)들을 동작시 I/O 스케줄링을 바꾸는데 사용하지 않았을까? 예를들어 다음과 같은 것이 있다. selectable I/O scheduler patch(http://lwn.net/Articles/50695/ : 밑에 원문을 붙여 놓을 것)는 sysfs attributes들을 완벽하게 I/O scheduler를 바꾸는데 사용할 수 있도록 해준다.

자 이제 몽땅 한군데로 넣어봅시다.

지금까지 연결되지 않은 각각의 모듈(조각)들을 봐 왔다. 전체 다이어그램은 밑에서 볼 수 있다(다른 문서에 넣던가 아니면 가로로 넣어야 할 듯: block-kobj.png, block-sysfs.png). 그리고, 밑의 sysfs에 대한 그림에서 각 kobject에 일치하는 sysfs 이름들을 볼 수 있다.
block-sysfs.png에서 보여지는 데이터들의 구조는 sysfs의 subtree인 /sys/block이 어떻게 구현되었는지 보여준다. 원래 전체 sysfs tree 구조는 이것보다 훨 복잡하다. /sys/block 밑에 보여지는 각 gendisk들은 현재 시스템에 있는 hardware를 보여주는 /sys/devices 밑에 각각의 엔트리로 존재하게 된다. 내부적으로 이 둘간의 링크는 gendisk structure의 driverfs_dev 필드가 담당한다. sysfs에서 둘간의 링크(block subsystem과 devices subsystem과의)는 두 하위 tree사이의 symbolic link로 표현된다.

다행히도 이 그림들은 sysfs tree와 device model data sturucture가 어떻게 구현이 되었고, 어떻게 연관이 되는지 알 수 있도록 도와준다. device model은 아주 복잡하다. 하지만, 한번 그 안에 숨겨진 개념을 잡게 되면, 전체에 대해 접근하는게 쉬워진다.


Legend:
Kobject parent
Kobject type
Kset membership
block_subsys
.kset
ktype_block
hda
hdb
Request Queue
hda1
hda2
hda3
hdaN
hdb1
hdb2
hdb3
hdbN
queue_ktype
ktype_part
Elevator
as_ktype


sysfs connections
Kobject attribute
block_subsys
.kset
ktype_block
/sys/block
/sys/block/hda
hda
/sys/block/hdb
hdb
/sys/block/hda/queue
Request Queue
/sys/block/hda/hda1 .../hda2 .../hda3 .../hdaN
hda1
hda2
hda3
hdaN
hdb1
hdb2
hdb3
hdbN
/sys/block/hda/queue/iosched
Elevator
queue_ktype
as_ktype
ktype_part

5. device driver hierarchy 초기화

여기서는 커널 시작서부터 추적을 하여 실제 커널의 device driver infrastructure가 어떻게 초기화 되는지 추적을 하였다. 중요한 부분은 driver hierarchy 초기화이기 때문에 그 부분에 대한 설명외에는 간단하게 처리하였다.

- init/main.c

```
asmlinkage void __init start_kernel(void)
{
...
      setup_arch(&command_line);
...
      rest_init();
}


static void noinline rest_init(void)
      __releases(kernel_lock)
{
      kernel_thread(init, NULL, CLONE_FS | CLONE_SIGHAND); // init kernel thread를 생성한다.
      numa_default_policy();
      unlock_kernel();
      preempt_enable_no_resched();

      /*
       * The boot idle thread must execute schedule()
       * at least one to get things moving:
       */
      schedule();                                    // schedule시작

      cpu_idle();
}


static int init(void * unused)
{
...
      /*
       * Do this before initcalls, because some drivers want to access
       * firmware files.
       */
      populate_rootfs();

      do_basic_setup();          // 커널에서 하는 일 치고는 커널 코어말고, 주변(driver등등)에 대한
                                 // 기본적인 초기화를 한다.

      /*
       * check if there is an early userspace init.  If yes, let it do all
       * the work
       */
      if (sys_access((const char __user *) "/init", 0) == 0)
            execute_command = "/init";
      else
            prepare_namespace();

      /*
       * Ok, we have completed the initial bootup, and
       * we're essentially up and running. Get rid of the
       * initmem segments and start the user-mode stuff..
       */
      free_initmem();
      unlock_kernel();
      system_state = SYSTEM_RUNNING;
      numa_default_policy();
```

```
        if (sys_open((const char __user *) "/dev/console", O_RDWR, 0) < 0)
                printk(KERN_WARNING "Warning: unable to open an initial console.\n");

        (void) sys_dup(0);
        (void) sys_dup(0);

        /*
         * We try each of these until one succeeds.
         *
         * The Bourne shell can be used instead of init if we are
         * trying to recover a really broken machine.
         */

        if (execute_command)
                run_init_process(execute_command);

        run_init_process("/sbin/init");                  //  /sbin/init process를 돌린다.
        run_init_process("/etc/init");
        run_init_process("/bin/init");
        run_init_process("/bin/sh");

        panic("No init found.  Try passing init= option to kernel.");
}



static void __init do_basic_setup(void)
{
        /* drivers will send hotplug events */
        init_workqueues();

        /*
         => kernel/kmod.c: "khelper"라는 이름의 workqueue생성, call_usermodehelper()함수를 돌려주기 위
한
           workq이다. 이 함수는 실제 /sbin/hotplug가 되겠지롱....hotplug관련 초기화
         */
        usermodehelper_init();

        /*
          => drivers/base/init.c, 드라이버 구조를 초기화 한다.
         여기서 platform, bus, 등등의 기본 초기화가 일어나고
         platform의 경우는 platform자체가 device로 등록이 된다.
         밑의 함수내용을 볼 것.
         */

        driver_init();

#ifdef CONFIG_SYSCTL
        sysctl_init();
#endif

        /* Networking initialization needs a process context */
        sock_init();

        /*
        architecture 처리 후, 예를 들어 setup_arch()에서 platform bus
        에 등록할 platform device들의 배열을 board란 ptr에 넣어 놓는다.
        그 후에 이 함수를 호출하면서 cpu.c의 s3c_arch_init()란 녀석이
        init section에 있으므로 호출을 해서 platform bus에 해당
        device들을 등록한다. 물론 그 전에 platform device와 bus가 생성
        되어 있어야 한다(그것은 바로 위의 driver_init()에서 이루어 진다)
        */

        do_initcalls();
}


- drivers/base/init.c
```

```
/**
 *     driver_init - initialize driver model.
 *
 *     Call the driver model init functions to initialize their
 *     subsystems. Called early from init/main.c.
 */

void __init driver_init(void)
{
      /* These are the core pieces */
      devices_init();
      buses_init();
      classes_init();
      firmware_init();

      /* These are also core pieces, but must come after the
       * core core pieces.
       */
      platform_bus_init();
      system_bus_init();
      cpu_dev_init();
      attribute_container_init();
}


- init/main.c
extern initcall_t __initcall_start[], __initcall_end[];

// 밑의 s3c_arch_init() 함수는 init section에 들어가게 되므로, 여기서 실행이 된다.
// 실행되면서 등록하고자 하는 platform device들을 platform_bus와 devices_subsystem에 등록하게 된다.
static void __init do_initcalls(void)
{
      initcall_t *call;
      int count = preempt_count();

      for (call = __initcall_start; call < __initcall_end; call++) {
            char *msg;

            if (initcall_debug) {
                  printk(KERN_DEBUG "Calling initcall 0x%p", *call);
                  print_fn_descriptor_symbol(": %s()", (unsigned long) *call);
                  printk("\n");
            }

            (*call)();

            msg = NULL;
            if (preempt_count() != count) {
                  msg = "preemption imbalance";
                  preempt_count() = count;
            }
            if (irqs_disabled()) {
                  msg = "disabled interrupts";
                  local_irq_enable();
            }
            if (msg) {
                  printk(KERN_WARNING "error in initcall at 0x%p: "
                        "returned with %s\n", *call, msg);
            }
      }

      /* Make sure there is no pending stuff from the initcall sequence */
      flush_scheduled_work();
}

- arch/arm/cpu.c
static int __init s3c_arch_init(void)
{
   int ret;
```

```
    // do the correct init for cpu

    printk("%s: %%%%%%%%%%%%%%%%%%%%%%%%%%%%\n", __FUNCTION__);


    if (cpu == NULL)
        panic("s3c_arch_init: NULL cpu\n");

    ret = (cpu->init)();
    if (ret != 0)
        return ret;

    if (board != NULL) {
        struct platform_device **ptr = board->devices;
        int i;

        for (i = 0; i < board->devices_count; i++, ptr++) {
            ret = platform_device_register(*ptr);

            if (ret) {
                printk(KERN_ERR "s3c24xx: failed to add board device %s (%d) @%p\n",
                                (*ptr)->name, ret, *ptr);
            }
        }

        /* mask any error, we may not need all these board
         * devices */
        ret = 0;
    }

    return ret;
}

arch_initcall(s3c_arch_init);
```

여러함수들이 실행이 되나 간단하게 platform초기와 쪽만 살펴본다. 이유는 특이하게 "platform" 이란 것도 device로 등록이 된다(bus만 등록이 되는게 아니라, device도 하나 등록이 되는게 좀 특이했음)

```
...

- drivers/base/platform.c

int __init platform_bus_init(void)
{
      gprintk("%s() device_register call\n", __FUNCTION__);
      device_register(&platform_bus);
      gprintk("%s() bus_register call\n", __FUNCTION__);
      return bus_register(&platform_bus_type);
}
...
```

6. Platform device registration

```
- arch/arm/mach-s3c2410/mach-aesop2440.c: platform device등록

static struct platform_device *aesop2440_devices[] __initdata = {
    &s3c_device_usb,
    &s3c_device_lcd,
    &s3c_device_bl,
    &s3c_device_wdt,
    &s3c_device_i2c,
```

```
    &s3c_device_iis,
    &s3c_device_sdi,
    &s3c_device_usbgadget,
    &s3c_device_ts,
    &s3c_device_nand,
    &s3c_device_sound,
    &s3c_device_buttons,
    &s3c_device_rtc,
};

static struct s3c24xx_board aesop2440_board __initdata = {
    .devices      = aesop2440_devices,
    .devices_count = ARRAY_SIZE(aesop2440_devices)
};

void __init aesop2440_map_io(void)
{
    s3c24xx_init_io(aesop2440_iodesc, ARRAY_SIZE(aesop2440_iodesc));

#ifdef CONFIG_S3C2440_INCLK12
    s3c24xx_init_clocks(12000000);
#else
    s3c24xx_init_clocks(16934400);
#endif
    s3c24xx_init_uarts(aesop2440_uartcfgs, ARRAY_SIZE(aesop2440_uartcfgs));

    // ==> 여기서 cpu.c의 s3c24xx_set_board()를 호출하여 static struct s3c24xx_board *board;
    // 의 board ptr로 등록한다.
    s3c24xx_set_board(&aesop2440_board);

    s3c_device_nand.dev.platform_data = &aesop_nand_info;
}



MACHINE_START(AESOP2440, "aESOP-2440")
    /* Maintainer: godori(ghcstop@gmail.com) www.aesop-embedded.org*/
    .phys_ram      = S3C2410_SDRAM_PA,
    .phys_io = S3C2410_PA_UART,
    .io_pg_offst    = (((u32)S3C24XX_VA_UART) >> 18) & 0xfffc,
    .boot_params    = S3C2410_SDRAM_PA + 0x100,

    .init_irq      = aesop2440_init_irq,
    .map_io        = aesop2440_map_io,              //==> 여기서 호출되고,
    .init_machine  = aesop2440_init,
    .timer         = &s3c24xx_timer,
MACHINE_END



- cpu.c

static struct s3c24xx_board *board;

void s3c24xx_set_board(struct s3c24xx_board *b)
{
       int i;

       board = b;

       if (b->clocks_count != 0) {
              struct clk **ptr = b->clocks;;

              for (i = b->clocks_count; i > 0; i--, ptr++)
                     s3c24xx_register_clock(*ptr);
       }
}
```

```
// 이 함수는 init/main.c의 do_basic_setup()에서 driver_init()이 호출되어서 platform bus가
// 등록이 된 후에 호출된다.
static int __init s3c_arch_init(void)
{
    int ret;

    // do the correct init for cpu

    if (cpu == NULL)
       panic("s3c_arch_init: NULL cpu\n");

    ret = (cpu->init)();
    if (ret != 0)
       return ret;

    if (board != NULL) {

        // 여기서 mach-xxxx.c에서 s3c24xx_set_board() 호출해서 등록이 된
        // device들을 platform_device로 등록한다.
       struct platform_device **ptr = board->devices;
       int i;

       for (i = 0; i < board->devices_count; i++, ptr++) {
             ret = platform_device_register(*ptr);

             if (ret) {
                   printk(KERN_ERR "s3c24xx: failed to add board device %s (%d) @%p\n", (*ptr)-
>name, ret, *ptr);
             }
       }

       /* mask any error, we may not need all these board
        * devices */
       ret = 0;
    }

    return ret;
}



- drivers/base/platform.c

/**
 *  platform_device_register - add a platform-level device
 *  @pdev:   platform device we're adding
 *
 */
int platform_device_register(struct platform_device * pdev)
{
    int i, ret = 0;

    if (!pdev)
       return -EINVAL;

    if (!pdev->dev.parent)
       pdev->dev.parent = &platform_bus; // struct device 형

    // 입력된 device의 bus type을 platform bus로 설정하고,
    pdev->dev.bus = &platform_bus_type;

    if (pdev->id != -1)
       snprintf(pdev->dev.bus_id, BUS_ID_SIZE, "%s.%u", pdev->name, pdev->id);
    else
       strlcpy(pdev->dev.bus_id, pdev->name, BUS_ID_SIZE);

    for (i = 0; i < pdev->num_resources; i++) {
       struct resource *p, *r = &pdev->resource[i];

       if (r->name == NULL)
```

```
                r->name = pdev->dev.bus_id;

        p = r->parent;
        if (!p) {
                if (r->flags & IORESOURCE_MEM)
                        p = &iomem_resource;
                else if (r->flags & IORESOURCE_IO)
                        p = &ioport_resource;
        }

        if (p && request_resource(p, r)) {
                printk(KERN_ERR
                       "%s: failed to claim resource %d\n",
                       pdev->dev.bus_id, i);
                ret = -EBUSY;
                goto failed;
        }
    }

    pr_debug("Registering platform device '%s'. Parent at %s\n",
        pdev->dev.bus_id, pdev->dev.parent->bus_id);


    // device를 등록한다. ==> drivers/base/core.c
    ret = device_register(&pdev->dev);
    if (ret == 0)
        return ret;

 failed:
    while (--i >= 0)
        if (pdev->resource[i].flags & (IORESOURCE_MEM|IORESOURCE_IO))
                release_resource(&pdev->resource[i]);
    return ret;
}
```

위의 코드에서 보면 cpu별 architecture platform device등록 함수인 s3c_arch_init()이 호출되면서 미리 등록해 놓았던 platform device들을 device_register()를 이용해서 등록이 된다.

나중에 mmc와 hotplug쪽의 설명을 보면 알 수 있겠지만, 일단, device_register()를 호출하게 되면 hotplug method가 호출이 된다. 이때 실제로 동작되는 hotplug함수는 device가 붙어 있는 bus의 hotplug함수이다. 즉, platform device가 등록되는 platform bus의 경우 hotplug함수가 NULL로 되어 있다. 해서 ***platform device의 경우 hotplug event가 발생하지 않는다.***

7. 등록되어 있는 subsystem

subsystem 등록은 위에서도 설명했듯이 다음과 같은 매크로를 이용해서 이루어진다.

```
#define decl_subsys(_name,_type,_hotplug_ops)
```

이 매크로를 이용해서 선언된 녀석들은 다음과 같다.

drivers/base에 선언된 subsystem을 살펴보면 다음과 같다.

```
root@aesopdev:/korea-dokdo/2613-telinfos/drivers/base# gref decl_subsys
./bus.c
133:decl_subsys(bus, &ktype_bus, NULL);

./sys.c
82:static decl_subsys(system, &ktype_sysdev, NULL);
```

```
./firmware.c
15:static decl_subsys(firmware, NULL, NULL);


// 이 경우 hotplug관련 함수들이 지정이 된다.
./core.c
161:decl_subsys(devices, &ktype_device, &device_hotplug_ops);

./class.c
74:static decl_subsys(class, &ktype_class, NULL);
407:static decl_subsys(class_obj, &ktype_class_device, &class_hotplug_ops);
```

이렇게 선언이 된 subsystem들은 subsystem_register()를 이용하여 최상위 subsystem이 된다 (sysfs top-level directory를
차지하게 된다.)

==> 이것은 drivers/base 디렉토리만 살펴본 것이고, 실제 drivers/block/genhd.c의 경우 block subsystem에 대한 선언이 존재한다.

```
root@aesopdev:/korea-dokdo/2613-telinfos/drivers/base# gref subsystem_register

./bus.c
586:    retval = subsystem_register(&bus->subsys);
637:    return subsystem_register(&bus_subsys);       //여기서 bus subsystem이 최상위로 등록


./sys.c
393:    return subsystem_register(&system_subsys);    //여기서 system subsystem이 최상위로 등록

./firmware.c
20:     return subsystem_register(s);
30:     return subsystem_register(&firmware_subsys);  //여기서 firmware subsystem이 최상위로 등록

./core.c
404:    return subsystem_register(&devices_subsys);   // devices....

./class.c
725:    retval = subsystem_register(&class_subsys);   // class
```

aesop-2440의 /sys를 보면

```
root@godori:~# l /sys
drwxr-xr-x   10 root    root          0 Jan  1 00:00 .
drwxr-xr-x   25 500     500        4096 Dec 16  2005 ..
drwxr-xr-x   43 root    root          0 Jan  1 00:00 block
drwxr-xr-x    8 root    root          0 Jan  1 00:00 bus
drwxr-xr-x   20 root    root          0 Jan  1 00:00 class
drwxr-xr-x    4 root    root          0 Jan  1 00:00 devices
drwxr-xr-x    2 root    root          0 Jan  1 00:00 firmware
drwxr-xr-x    2 root    root          0 Jan  1 00:00 kernel
drwxr-xr-x    9 root    root          0 Jan  1 00:00 module
drwxr-xr-x    2 root    root          0 Jan  1 00:00 power
```

이렇게 되어 있다. 실제로 여기서 block의 경우는 원래는 class에 속하게 작성되어야 하나전통적인 이유로 block subsystem을 구성하게 되었다는 말이 있다(ldd 3rd edition chapter 14). class는 다른 subsystem에 등록이 되어 있더라도 비슷한 종류의 device들의 구분이기 때문이다.

그리고, subsystem의 경우 실제로 sysfs에 나타날수도 잇고 안나타날 수도 있다는 설명이 있다.

8. bus registration

버스는 리눅스 driver model의 application개념이다. 즉, driver model은 각 object들이 그냥 존재해 있다고 봐도 되는 상태이고, 그것을 이용해서 시스템에서 사용할 수 있도록 만들어주는 가교 역할을 하는게 bus이다. bus는 실제 물리적인 버스를 bus structure로 구성할 수도 있고(추상화시킴), 가상적인 bus를 만들어서 사용할 수도 있다.

물리적인 bus를 구성하는 예는 PCI, USB, MMC등과 같은 버스의 경우이며, 가상적인 bus의 경우는 시스템에 특정 버스에 붙지 않지만, 존재하는 device들을 붙일 때 사용되는 platform bus를 말한다.

```
- device.h
struct bus_type {
       const char            * name;

       struct subsystem       subsys;
       struct kset           drivers;
       struct kset           devices;
       struct klist          klist_devices;
       struct klist          klist_drivers;

       struct bus_attribute    * bus_attrs;
       struct device_attribute * dev_attrs;
       struct driver_attribute * drv_attrs;

       /* 새로운 디바이스나 드라이버를 이 버스에 등록할때 여러번 호출이 가능하다.
       이 함수는 주어진 디바이스를 주어진 driver가 처리할 수 있다면 0이 아닌 값을 반환한다.)
       int           (*match)(struct device * dev, struct device_driver * drv);
       int           (*hotplug) (struct device *dev, char **envp,
                                 int num_envp, char *buffer, int buffer_size);
       int           (*suspend)(struct device * dev, pm_message_t state);
       int           (*resume)(struct device * dev);
};



- bus.c

/**
 *  bus_register - register a bus with the system.
 *  @bus:    bus.
 *
 *  Once we have that, we registered the bus with the kobject
 *  infrastructure, then register the children subsystems it has:
 *  the devices and drivers that belong to the bus.
 */
int bus_register(struct bus_type * bus)
{
    int retval;

    /* 입력된 bus의 subsys.kset.kobj에 이름을 입력한다. */
    retval = kobject_set_name(&bus->subsys.kset.kobj, "%s", bus->name);
    if (retval)
       goto out;

    /**

    bus_subsys는 decl_subsys라는 매크로를 가지고 만들어지 녀석으로

    #define decl_subsys(_name,_type,_hotplug_ops) \
    struct subsystem _name##_subsys = { \
```

```
    .kset = { \
        .kobj = { .name = __stringify(_name) }, \
        .ktype = _type, \
        .hotplug_ops =_hotplug_ops, \
    } \
}

// bus_subsys는 이렇게 만들어진 것이다.
static struct kobj_type ktype_bus = {
    .sysfs_ops      = &bus_sysfs_ops,
}   ;

decl_subsys(bus, &ktype_bus, NULL);
```

즉, 위의 매크로를 이용해서 만들어진 bus_subsys는 다음과 같이 선언된 것이다.

```
struct subsystem bus_subsys = {
    .kset = {
        .kobj = { .name = "bus" },
        .ktype = ktype_bus,
        .hotplug_ops = NULL,
    }
}



// kobject.h
*      subsys_set_kset(obj,subsys) - set kset for subsystem
*      @obj:          ptr to some object type.
*      @subsys:       a subsystem object (not a ptr).
*
*      Can be used for any object type with an embedded ->subsys.
*      Sets the kset of @obj's kobject to @subsys.kset. This makes
*      the object a member of that kset.
```

입력된 obj의 kobject의 kset에 subsys.kset로 설정을 한다.

```
#define subsys_set_kset(obj,_subsys) \
    (obj)->subsys.kset.kobj.kset = &(_subsys).kset
```

이 경우는 bus register를 하는 bus라는 bus_type변수에 bus_subsys의
kset을 설정한다.

즉, 등록되는 버스에 bus subsystem 이라는 공통점을 주기 위한 설정이다.
왜냐하면 kset은 공통적인 것들을 표시하는 것이기 때문이다.

입력된 bus가 bus subsystem이라는 하나의 종류(kset)에 속한다고 표시하는 것이다.

```
----------
입력된 버스의 subsys.kset.kobj.kset으로 bus_subsys의 kset이 설정된 것이다.
해서 이 bus는 "bus"란 서브시스템으로 붙게 된 것이다.
*/

subsys_set_kset(bus, bus_subsys);  // bus_subsys

// bus->subsys의 kset을 초기화 시키고(kobject_init()까지만 한다. add는 나중에 하나?),
// bus->subsys의 kset을 kset에 추가시킨다.
retval = subsystem_register(&bus->subsys);
if (retval)
   goto out;

kobject_set_name(&bus->devices.kobj, "devices");
bus->devices.subsys = &bus->subsys;
retval = kset_register(&bus->devices);
if (retval)
   goto bus_devices_fail;

kobject_set_name(&bus->drivers.kobj, "drivers");
bus->drivers.subsys = &bus->subsys;
```

```
    bus->drivers.ktype = &ktype_driver;
    retval = kset_register(&bus->drivers);
    if (retval)
       goto bus_drivers_fail;

    klist_init(&bus->klist_devices);
    klist_init(&bus->klist_drivers);
    bus_add_attrs(bus);

    pr_debug("bus type '%s' registered\n", bus->name);
    return 0;

bus_drivers_fail:
    kset_unregister(&bus->devices);
bus_devices_fail:
    subsystem_unregister(&bus->subsys);
out:
    return retval;
}
```

bus란 녀석의 subsys 멤버인 kset.kobj.kset을 bus_subsystem으로 등록하고, 자기의 subsys른 초기화한후 해당 kset을 등록시키면 bus란 녀석은 bus_subsystem에 속한 subsystem이 되는 것이다 (물론 subsystem_register를 호출한 후).

그 다음에 device와 drive kset을 만들고, 해당 kset들의 subsys로 위에서 등록된 bus->subsys로 등록한다면 bus는 bus_subsys에 속한 넘이 되고, device와 driver kset은 결국 bus_subsys에 속하게 되는 것이다.

```
- kobject.c
/**
 *     subsystem_register - register a subsystem.
 *     @s:    the subsystem we're registering.
 *
 *     Once we register the subsystem, we want to make sure that
 *     the kset points back to this subsystem for correct usage of
 *     the rwsem.
 */

int subsystem_register(struct subsystem * s)
{
       int error;

       subsystem_init(s);
       pr_debug("subsystem %s: registering\n",s->kset.kobj.name);

      // 그냥 subsystem의 kset의 뒤에 add를 할 뿐이다. bus등록일 경우는 bus라는 subsystem에 add하는 것이
다.
       if (!(error = kset_add(&s->kset))) {
              if (!s->kset.subsys)
                     s->kset.subsys = s;
       }
       return error;
}
```

bus와 subsystem과의 관계를 여기서 생각해 보자. subsystem이란 녀석은 kernel model의 가장 상위 레벨 개념이다. 즉, top쪽의 개념이고, bus는 그 subsystem중 하나인 bus subsystem에 속하는 것이다.

subsystem은 아주 추상적인 커널 모델의 상위 개념이고 bus는 그 subsystem중 하나를 나타내며 (즉, bus란 녀석은 (실제적인)device와 device에 대해서는 맨 상위 개념이다) 실제적인 device와

driver들이 들러붙어 있는 상위 개념이 된다.

bus란 녀석이 들어오면 입력된 버스는 bus_subsystem이란 녀석으로 속하게 된다. 즉, 속한다고 얘기하기 보다는 같은 특성이라고 표시되는 같은 kset이 설정이 된다(즉, 너는 bus이니 당연히 bus_subsystem으로 들어가라...고 하는).

집합개념인데 bus와 subsystem은 완전히 subsystem이 bus를 포함하는 형태가 아니라 서로의 교집합을
가지고 있는 형태로 생각하면 되겠다(kernel driver model에서의 kset의 개념으로 보면 속한다고 봐야하지만 시스템적인 입장에서보면 교집합을 가진 형태이다)

즉, bus는 bus나름대로의 모양이 있다.(위의 스트럭쳐를 보면 알 수 있듯이 subsystem말고도, device라는 kset, driver라는 kset등을 가지고 있다.) 하지만 bus는 bus_subsystem에 하나의 줄을 매달고 있듯이
서로 연결되어 있다. 하지만 종속적으로 생각하면 subsystem의 일부인 bus_subsystem에 bus가 자식으로 매달려 있는 것이다.

※ 버스에 대한것은 더 추가해서 작성할 것.

9. device register

여기서는 device를 등록하는 코드를 살펴본다. device를 등록하는 것은 1장의 전체 그림에서도 볼 수 있듯이 kernel driver model에서는 devices_subsystem의 kset으로 해당 device kobject의 kset을 설정하는, 즉, devices_subsystem의 kset에 속하게 된다고 볼 수 있다.
반면에 system적인 입장에서는 해당 device는 소속 bus에도 등록이 되고, 등록되면서 매칭되는 driver가 있나 검색을 하고, 매칭이 되는 녀석이 있을 경우 driver->probe()함수를 호출하고 성공시 서로 bind를 하는 것을 볼 수 있다. 자세한 내용은 코드에 대한 설명을 보면 나와있으니 밑을 보기 바란다.

여기서 눈여겨 봐야할 kernel driver model hierarch관련 내용은 kobject_add()과 관련해서 kobject를 kset에 추가하는 방법은 kobject_add()를 호출하기 전에 kobject가 자기랑 상관있다고 생각되는 kset을 설정해야 한다는데 있다.

```
- drivers/base/core.c

/**
 *     device_register - register a device with the system.
 *     @dev:   pointer to the device structure
 *
 *     This happens in two clean steps - initialize the device
 *     and add it to the system. The two steps can be called
 *     separately, but this is the easiest and most common.
 *     I.e. you should only call the two helpers separately if
 *     have a clearly defined need to use and refcount the device
 *     before it is added to the hierarchy.
 */

int device_register(struct device *dev)
{
```

```
        device_initialize(dev);
        return device_add(dev);
}



/**
 *     device_initialize - init device structure.
 *     @dev:   device.
 *
 *     This prepares the device for use by other layers,
 *     including adding it to the device hierarchy.
 *     It is the first half of device_register(), if called by
 *     that, though it can also be called separately, so one
 *     may use @dev's fields (e.g. the refcount).
 */

void device_initialize(struct device *dev)
{

    /**
    kobject.h

    *     kobj_set_kset_s(obj,subsys) - set kset for embedded kobject.
    *     @obj:          ptr to some object type.
    *     @subsys:       a subsystem object (not a ptr).
    *
    *     Can be used for any object type with an embedded ->kobj.

    kobj.kset을 subsys.kset으로 할당하는 것이다.

    #define kobj_set_kset_s(obj,subsys) \
        (obj)->kobj.kset = &(subsys).kset


    현재 입력된 device의 kset이 속한 subsystem이 devices_subsystem이라고 설정하는 것이다.
    */

    kobj_set_kset_s(dev, devices_subsys);
    kobject_init(&dev->kobj);
    klist_init(&dev->klist_children);
    INIT_LIST_HEAD(&dev->dma_pools);
    init_MUTEX(&dev->sem);
}


/**
 *  device_add - add device to device hierarchy.
 *  @dev:    device.
 *
 *  This is part 2 of device_register(), though may be called
 *  separately _iff_ device_initialize() has been called separately.
 *
 *  This adds it to the kobject hierarchy via kobject_add(), adds it
 *  to the global and sibling lists for the device, then
 *  adds it to the other relevant subsystems of the driver model.
 */
int device_add(struct device *dev)
{
    struct device *parent = NULL;
    int error = -EINVAL;

    dev = get_device(dev);
    if (!dev || !strlen(dev->bus_id))
        goto Error;

    parent = get_device(dev->parent);

    pr_debug("DEV: registering device: ID = '%s'\n", dev->bus_id);
```

```
    /* first, register with generic layer. */
    kobject_set_name(&dev->kobj, "%s", dev->bus_id);
    if (parent)
       dev->kobj.parent = &parent->kobj;

// 여기서 kobject를 kset에 등록한다. kset은 devices_subsystem의 kset으로 이미 위에서 지정이 되었다.
    if ((error = kobject_add(&dev->kobj)))
       goto Error;
    kobject_hotplug(&dev->kobj, KOBJ_ADD);
    if ((error = device_pm_add(dev)))
       goto PMError;
    if ((error = bus_add_device(dev))) // 여기서 device를 bus에 붙인다.
       goto BusError;
    if (parent)
       klist_add_tail(&parent->klist_children, &dev->knode_parent);

    /* notify platform of device entry */
    if (platform_notify)
       platform_notify(dev);
 Done:
    put_device(dev);
    return error;
 BusError:
    device_pm_remove(dev);
 PMError:
    kobject_hotplug(&dev->kobj, KOBJ_REMOVE);
    kobject_del(&dev->kobj);
 Error:
    if (parent)
       put_device(parent);
    goto Done;
}
```

device_register와 driver_register와는 엄연히 다른 것이다. 가장 큰 차이는 등록이 되는 kset이 틀리다는 것이다. device는 devices_subsystem에 등록이 되고, driver는 bus의 driver관련 kset에 등록이 되는 것이다. 이 차이의 가장 큰 점은 device_register에서는 device에 대한 hotplug처리가 된다는데 있다. 왜냐하면 hotplug는 kset에 종속되기 때문이다(물론, driver가 속한 bus의 driver관리 kset에 hotplug가 등록이 되어 있다면 hotplug가 일어난다).

platform_device_register()는 platform device관련 부분을 보면 된다.

```
- bus.c
/**
 *     bus_add_device - add device to bus
 *     @dev:   device being added
 *
 *     - Add the device to its bus's list of devices.
 *     - Try to attach to driver.
 *     - Create link to device's physical location.
 */
int bus_add_device(struct device * dev)
{
       struct bus_type * bus = get_bus(dev->bus);
       int error = 0;

       if (bus) {
              pr_debug("bus %s: add device %s\n", bus->name, dev->bus_id);
              device_attach(dev);
              klist_add_tail(&bus->klist_devices, &dev->knode_bus);
              error = device_add_attrs(bus, dev);
              if (!error) {
                     sysfs_create_link(&bus->devices.kobj, &dev->kobj, dev->bus_id);
                     sysfs_create_link(&dev->kobj, &dev->bus->subsys.kset.kobj, "bus");
              }
```

```
        }
        return error;
}


- dd.c
/**
 *      device_attach - try to attach device to a driver.
 *      @dev:   device.
 *
 *      Walk the list of drivers that the bus has and call
 *      driver_probe_device() for each pair. If a compatible
 *      pair is found, break out and return.
 *
 *      Returns 1 if the device was bound to a driver;
 *      0 if no matching device was found; error code otherwise.
 */
int device_attach(struct device * dev)
{
        int ret = 0;

        down(&dev->sem);
        if (dev->driver) {
                device_bind_driver(dev);
                ret = 1;
        } else
                ret = bus_for_each_drv(dev->bus, NULL, dev, __device_attach);
        up(&dev->sem);
        return ret;
}


static int __device_attach(struct device_driver * drv, void * data)
{
        struct device * dev = data;
        return driver_probe_device(drv, dev);
}


// driver_register시에도 결국은 마지막으로 이 함수를 호출한다.
/**
 *  driver_probe_device - attempt to bind device & driver.
 *  @drv:    driver.
 *  @dev:    device.
 *
 *  First, we call the bus's match function, if one present, which
 *  should compare the device IDs the driver supports with the
 *  device IDs of the device. Note we don't do this ourselves
 *  because we don't know the format of the ID structures, nor what
 *  is to be considered a match and what is not.
 *
 *
 *  This function returns 1 if a match is found, an error if one
 *  occurs (that is not -ENODEV or -ENXIO), and 0 otherwise.
 *
 *  This function must be called with @dev->sem held.
 */
int driver_probe_device(struct device_driver * drv, struct device * dev)
{
    int ret = 0;

    if (drv->bus->match && !drv->bus->match(dev, drv))
       goto Done;

    pr_debug("%s: Matched Device %s with Driver %s\n",
        drv->bus->name, dev->bus_id, drv->name);
    dev->driver = drv;
    if (drv->probe) {
       ret = drv->probe(dev);  // 여기서 매칭이 되는 driver의 probe 함수를 호출한다.
       if (ret) {
```

```
            dev->driver = NULL;
            goto ProbeFailed;
      }
   }
   device_bind_driver(dev);
   ret = 1;
   pr_debug("%s: Bound Device %s to Driver %s\n",
       drv->bus->name, dev->bus_id, drv->name);
   goto Done;

ProbeFailed:
   if (ret == -ENODEV || ret == -ENXIO) {
      /* Driver matched, but didn't support device
       * or device not found.
       * Not an error; keep going.
       */
      ret = 0;
   } else {
      /* driver matched but the probe failed */
      printk(KERN_WARNING
            "%s: probe of %s failed with error %d\n",
            drv->name, dev->bus_id, ret);
   }
Done:
   return ret;
}

/**
 *     device_bind_driver - bind a driver to one device.
 *     @dev:   device.
 *
 *     Allow manual attachment of a driver to a device.
 *     Caller must have already set @dev->driver.
 *
 *     Note that this does not modify the bus reference count
 *     nor take the bus's rwsem. Please verify those are accounted
 *     for before calling this. (It is ok to call with no other effort
 *     from a driver's probe() method.)
 *
 *     This function must be called with @dev->sem held.
 */
void device_bind_driver(struct device * dev)
{
       pr_debug("bound device '%s' to driver '%s'\n",
              dev->bus_id, dev->driver->name);
       klist_add_tail(&dev->driver->klist_devices, &dev->knode_driver);

       // sysfs에서 심볼릭 링크를 건다. device와 driver사이를
       // fs/sysfs/symlink.c
       sysfs_create_link(&dev->driver->kobj, &dev->kobj,
                      kobject_name(&dev->kobj));
       sysfs_create_link(&dev->kobj, &dev->driver->kobj, "driver");
}
```

driver_probe_device()함수를 호출해서 device_register시 혹은 driver_register 호출시 등록된 device와 driver를 서로 비교해서 먼저 등록이 되어 있는 경우 검사를 해서 알맞은게 있으면 driver를 probe하고, 서로 bind를 시킨다.

이러한 일은 driver_register()를 추적해도 마찬가지로 발생한다.

10. driver register

드라이버의 등록은 각 모든 dirver에서 이루어 진다. 2.4.x대의 커널의 경우는 module을 초기화만 해주면 driver가 등록이 되었다. 물론, 2.6.x대의 커널에서도 모듈 초기화만 해주면 드라이버를 동작시킬 수는 있으나, 커널 driver model hierarchy에는 등록이 되지 않는다. 해서 sysfs와 udev등과 같은 메카니즘을 사용하지 못한다.

2.6 커널 고유의 driver model을 사용하고, sysfs등과 같은 userspace mechanism을 사용하고자 할 경우는 반드시 kernel driver hierarchy에 등록을 시켜야 한다.

그 등록 절차는 driver_register()를 호출하면서부터 시작된다. driver는 kernel driver model과의 관계보다 시스템적인 측면에 더 가까운

```
- drivers/base/driver.c
/**
 *     driver_register - register driver with bus
 *     @drv:   driver to register
 *
 *     We pass off most of the work to the bus_add_driver() call,
 *     since most of the things we have to do deal with the bus
 *     structures.
 *
 *     The one interesting aspect is that we setup @drv->unloaded
 *     as a completion that gets complete when the driver reference
 *     count reaches 0.
 */
int driver_register(struct device_driver * drv)
{
       klist_init(&drv->klist_devices);
       init_completion(&drv->unloaded);
       return bus_add_driver(drv);
}


- include/linux/klist.h

/*
 *     klist.h - Some generic list helpers, extending struct list_head a bit.
 *
 *     Implementations are found in lib/klist.c
 */

struct klist {
       spinlock_t            k_lock;
       struct list_head       k_list;
};


- lib/klist.c

/**
 *     klist_init - Initialize a klist structure.
 *     @k:    The klist we're initializing.
 */

void klist_init(struct klist * k)
{
       INIT_LIST_HEAD(&k->k_list);
       spin_lock_init(&k->k_lock);
}


- device.h
struct bus_type {
```

```
        const char              * name;

        struct subsystem        subsys;
        struct kset             drivers;
        struct kset             devices;
        struct klist            klist_devices;
        struct klist            klist_drivers;

        struct bus_attribute    * bus_attrs;
        struct device_attribute * dev_attrs;
        struct driver_attribute * drv_attrs;

        int             (*match)(struct device * dev, struct device_driver * drv);
        int             (*hotplug) (struct device *dev, char **envp,
                                    int num_envp, char *buffer, int buffer_size);
        int             (*suspend)(struct device * dev, pm_message_t state);
        int             (*resume)(struct device * dev);
};
```

- bus.c

```
/**
 *  bus_add_driver - Add a driver to the bus.
 *  @drv:    driver.
 *
 */
int bus_add_driver(struct device_driver * drv)
{
    struct bus_type * bus = get_bus(drv->bus);
    int error = 0;

    if (bus) {
       pr_debug("bus %s: add driver %s\n", bus->name, drv->name);
       error = kobject_set_name(&drv->kobj, "%s", drv->name);
       if (error) {
             put_bus(bus);
             return error;
       }
       drv->kobj.kset = &bus->drivers;  // bus의 drivers란 kset을 이 driver의 kset으로 지정

       // kobject등록, hotplug event도 일어나나 실제 kset에 hotplug가 없을 경우 동작하지 않는다.
       if ((error = kobject_register(&drv->kobj))) {
             put_bus(bus);
             return error;
       }

       driver_attach(drv);
       klist_add_tail(&bus->klist_drivers, &drv->knode_bus);
       module_add_driver(drv->owner, drv);

       driver_add_attrs(bus, drv);
       driver_create_file(drv, &driver_attr_unbind);
       driver_create_file(drv, &driver_attr_bind);
    }
    return error;
}
```

- dd.c

```
/**
 *     driver_attach - try to bind driver to devices.
 *     @drv:   driver.
 *
 *     Walk the list of devices that the bus has on it and try to
 *     match the driver with each one.  If driver_probe_device()
 *     returns 0 and the @dev->driver is set, we've found a
 *     compatible pair.
 */
```

```
void driver_attach(struct device_driver * drv)
{
	bus_for_each_dev(drv->bus, NULL, drv, __driver_attach);
}


static int __driver_attach(struct device * dev, void * data)
{
	struct device_driver * drv = data;

	/*
	 * Lock device and try to bind to it. We drop the error
	 * here and always return 0, because we need to keep trying
	 * to bind to devices and some drivers will return an error
	 * simply if it didn't support the device.
	 *
	 * driver_probe_device() will spit a warning if there
	 * is an error.
	 */

	down(&dev->sem);
	if (!dev->driver)
		driver_probe_device(drv, dev);
	up(&dev->sem);


	return 0;
}


// device_register시에도 결국은 마지막으로 이 함수를 호출한다.
/**
 *  driver_probe_device - attempt to bind device & driver.
 *  @drv:    driver.
 *  @dev:    device.
 *
 *  First, we call the bus's match function, if one present, which
 *  should compare the device IDs the driver supports with the
 *  device IDs of the device. Note we don't do this ourselves
 *  because we don't know the format of the ID structures, nor what
 *  is to be considered a match and what is not.
 *
 *
 *  This function returns 1 if a match is found, an error if one
 *  occurs (that is not -ENODEV or -ENXIO), and 0 otherwise.
 *
 *  This function must be called with @dev->sem held.
 */
int driver_probe_device(struct device_driver * drv, struct device * dev)
{
    int ret = 0;

    if (drv->bus->match && !drv->bus->match(dev, drv))
       goto Done;

    pr_debug("%s: Matched Device %s with Driver %s\n",
        drv->bus->name, dev->bus_id, drv->name);
    dev->driver = drv;
    if (drv->probe) {
       ret = drv->probe(dev);  // 여기서 probe 함수를 호출한다.
       if (ret) {
             dev->driver = NULL;
             goto ProbeFailed;
       }
    }
    device_bind_driver(dev);    // device와 driver를 bind한다.
    ret = 1;
    pr_debug("%s: Bound Device %s to Driver %s\n",
        drv->bus->name, dev->bus_id, drv->name);
    goto Done;
```

```
 ProbeFailed:
    if (ret == -ENODEV || ret == -ENXIO) {
       /* Driver matched, but didn't support device
        * or device not found.
        * Not an error; keep going.
        */
       ret = 0;
    } else {
       /* driver matched but the probe failed */
       printk(KERN_WARNING
             "%s: probe of %s failed with error %d\n",
             drv->name, dev->bus_id, ret);
    }
 Done:
    return ret;
}



/**
 *     device_bind_driver - bind a driver to one device.
 *     @dev:   device.
 *
 *     Allow manual attachment of a driver to a device.
 *     Caller must have already set @dev->driver.
 *
 *     Note that this does not modify the bus reference count
 *     nor take the bus's rwsem. Please verify those are accounted
 *     for before calling this. (It is ok to call with no other effort
 *     from a driver's probe() method.)
 *
 *     This function must be called with @dev->sem held.
 */
void device_bind_driver(struct device * dev)
{
       pr_debug("bound device '%s' to driver '%s'\n",
              dev->bus_id, dev->driver->name);
       klist_add_tail(&dev->driver->klist_devices, &dev->knode_driver);

       // device와 driver사이에 sysfs에서 심볼릭 링크를 건다.
       // fs/sysfs/symlink.c
       sysfs_create_link(&dev->driver->kobj, &dev->kobj,
                         kobject_name(&dev->kobj));
       sysfs_create_link(&dev->kobj, &dev->driver->kobj, "driver");
}
```

driver_probe_device()함수를 호출해서 device_register시 혹은 driver_register 호출시 등록된 device와 driver를 서로 비교해서 먼저 등록이 되어 있는 경우 검사를 해서 알맞은게 있으면 driver를 probe하고, 서로 bind를 시킨다.

이러한 일은 device_register()를 추적해도 마찬가지로 발생한다.

11. Class & MMC/SD bus, hotplug code flow

이 장에서는 위에서 지금까지 봐왔던 kernel의 driver model쪽의 입장에서 보면 관련 object의 등록에 대한 것을 예를 들고, 하나의 장비를 구성하는 system적인 입장에서 보는 bus, class등에 대한 설명과 예를 보게 된다.

11.1. Class

class 자체로도 하나의 chapter를 차지할 정도가 되지만, 워낙 커널 driver model관련 코드가 복잡하고 게다가 하나의 독립적인 존재처럼 되어 있기 때문에 이 문서에서는 개념과 코드의 flow에 곁들여 약간의 설명을 하고 넘어간다.
위의 subsystem쪽 예에서 볼 수 있듯이 class란 녀석은 하나의 subsystem으로 구성이 된다. 즉 class subsystem이 따로 존재한다는 말이다.

class는 디바이스를 분류할때 소속에 따라 분류하는 개념이 아니라(ex> 어느 버스에 몇번 device와 같은) 디바이스의 기능으로 분류는 하는것이다. 예를 들어 SCSI disk나 ATA disk의 경우 소속은 서로 틀리지만, 기능상으로는 다 같은 디스크인 block device일 뿐이다.

거의 모든 클래스는 sysfs에서 /sys/class 밑에 나타나게 되어 있다. 예로 모든 네트워크 인터페이스는 그것의 물리적인 소속에 관계없이 /sys/class/net 아래 들어가게 된다. 비슷하게 입력 디바이스는 /sys/class/input에서 찾을 수 있다. 하지만 block device들의 경우 역사적인 이유로 /sys/block에 따로 subsystem을 만들어서 존재하게 된다.

이건 아마도 block device hierachy쪽을 보면 알 수 있듯이 너무나 복잡하기 때문에 따로 subsystem을 만들어 놓은 것 같다. 다른 디바이스들은 그냥 디바이스 하나로 존재하는데 block device들의 경우 partition등과 같은 복잡한 것이 서브로 다시 존재하기 때문이다.

class관련 함수들은 커널 2.6.13 버전서부터 약간의 변화가 있었다. 자세한 내용은 클래스 관련 소스와 Documentation/driver-model 쪽의 class.txt를 참조할 것.

※ 여기서는 class는 그냥 거쳐가는 정도로만 인식하고자 하는데, class를 다루는데 가장 중요한 개념은 하나의 디바이스가 존재할때 시스템 측면에서 보는 bus와 커널 드라이버 모델측면에서 보는 subsystem, kset, kobject와는 다른 class측면도 존재한다는 것이다. 즉, 하나의 디바이스를 버스에 등록하고, subsystem에 등록할때, class에도 자동으로 등록이 되는것이 아니라 직접 class 관련함수들을 가지고 class subsystem에 등록을 해 줘야 한다는 것이다.

11.2. devices subsystem과 hotplug

리눅스가 올라간 시스템에 등록되는 모든 device는 devices_subsystem에 등록이 된다(내가 잘 못 알고 있을지 모르지만, 지금까지 파악한 결과로는 이렇다...^^)

```
- drivers/base/core.c

static int dev_hotplug(struct kset *kset, struct kobject *kobj, char **envp,
                       int num_envp, char *buffer, int buffer_size)
{
...

       if (dev->bus->hotplug) {
              /* have the bus specific function add its stuff */

              // mmc의 경우 mmc_bus_hotplug가 호출된다.
              retval = dev->bus->hotplug (dev, envp, num_envp, buffer, buffer_size);
                     if (retval) {
                     pr_debug ("%s - hotplug() returned %d\n",
                             __FUNCTION__, retval);
```

```
                }
        }

        return retval;
}

static struct kset_hotplug_ops device_hotplug_ops = {
        .filter =       dev_hotplug_filter,
        .name =         dev_hotplug_name,
        .hotplug =      dev_hotplug,
};

/**
 *      device_subsys - structure to be registered with kobject core.
 */

decl_subsys(devices, &ktype_device, &device_hotplug_ops); //devices_subsys라는 subsystem을 선언한
다.
```

11.3. MMC/SD bus와 driver등록

위의 코드에서 dev->bus->hotplug는 mmc의 경우 mmc_sysfs.c의

```
static int
mmc_bus_hotplug(struct device *dev, char **envp, int num_envp, char *buf,
                int buf_size)
{
...
}
```

를 호출하게 되고, 이녀석은

```
static struct bus_type mmc_bus_type = {
        .name           = "mmc",
        .dev_attrs      = mmc_dev_attrs,
        .match          = mmc_bus_match,
        .hotplug        = mmc_bus_hotplug,
        .suspend        = mmc_bus_suspend,
        .resume         = mmc_bus_resume,
};
```

에서 지정이 되어 있으며

모듈 초기화시 mmc bus를 bus_register를 이용해서 등록을 한다. 이 때 해당 버스에 대한 hotplug 계열 함수들도 등록이 되지롱...^^

```
static int __init mmc_init(void)
{
        return bus_register(&mmc_bus_type);
}

static void __exit mmc_exit(void)
{
        bus_unregister(&mmc_bus_type);
}
```

이렇게 버스를 등록한 후 mmc block device나 혹은 다른(SDIO등과 같은) 드라이버가 mmc bus에 등록이 될 경우는

```
/**
```

```
 *     mmc_register_driver - register a media driver
 *     @drv: MMC media driver
 */
int mmc_register_driver(struct mmc_driver *drv)
{
       drv->drv.bus = &mmc_bus_type;
       drv->drv.probe = mmc_drv_probe;
       drv->drv.remove = mmc_drv_remove;
       return driver_register(&drv->drv);
}

EXPORT_SYMBOL(mmc_register_driver);

/**
 *     mmc_unregister_driver - unregister a media driver
 *     @drv: MMC media driver
 */
void mmc_unregister_driver(struct mmc_driver *drv)
{
       drv->drv.bus = &mmc_bus_type;
       driver_unregister(&drv->drv);
}
```

이런식으로 driver 등록하는 함수를 만들어서 이 드라이버의 버스로 mmc_bus를 지정하고 등록을 한다.

이 녀석을 호출하는 예제로 block driver의 경우 mmc_blk.c에 있는데

```
static struct mmc_driver mmc_driver = {
       .drv          = {
              .name   = "mmcblk",
       },
       .probe        = mmc_blk_probe,
       .remove       = mmc_blk_remove,
       .suspend      = mmc_blk_suspend,
       .resume       = mmc_blk_resume,
};
```

로 해서 block driver를 등록한 후 sysfs의 드라이버에도 등록을 하는 것이다.

```
static int __init mmc_blk_init(void)
{
       int res = -ENOMEM;

       res = register_blkdev(major, "mmc");
       if (res < 0) {
              printk(KERN_WARNING "Unable to get major %d for MMC media: %d\n",
                     major, res);
              goto out;
       }
       if (major == 0)
              major = res;

       devfs_mk_dir("mmc");
       return mmc_register_driver(&mmc_driver);

 out:
       return res;
}

static void __exit mmc_blk_exit(void)
{
       mmc_unregister_driver(&mmc_driver);
       devfs_remove("mmc");
       unregister_blkdev(major, "mmc");
}
```

block driver의 경우 block subsystem에도 등록이 되고(register_blkdev()를 통해서), mmc_register_driver()를 통해서 mmc의 driver로도 등록된다.

여기서 주의할 점은 block subsystem에 등록이 될 때는 device로 등록이 되는 것이고, mmc bus에 등록이 될때는 block driver로 등록이 된다는 것이다.

11.4. MMC/SD card insertion & hotplug

mmc/sd card의 경우 컨넥터에 카드를 삽입하면 device가 생성이 되고 이 생성된 device는 device_subsystem에 붙어서 hotplug동작을 한다.

class device로 등록이 되는 platform device인 mmc/sd controller(s3c24x0의 경우)에 대한 hotplug의 경우는 밑의 class에 관련된 동작을 하게 된다.

```
- mmc_sysfs.c

/*
 * Internal function.  Initialise a MMC card structure.
 */
void mmc_init_card(struct mmc_card *card, struct mmc_host *host)
{
    memset(card, 0, sizeof(struct mmc_card));
    card->host = host;

    /*
     * device에 대한 kobject초기화, 이 kobj의 kset으로 devices_subsys의 kset을 대입한다.
     */
    device_initialize(&card->dev);
    card->dev.parent = card->host->dev;

    /*
     * 여기서 card의 device의 bus에 mmc_bus_type이 대입된다.
     * bus의 hotplug쪽도 같이 대입되는 것이다.
     */
    card->dev.bus = &mmc_bus_type;

    card->dev.release = mmc_release_card;
}
```

이시점에서 device를 kobject로 등록하면서 kobject의 kset pointer로 devices_subsystem kset의 주소를 설정 한다. 해서 이 device는 devices_subsystem의 일원이 된 것이다.

```
/*
 * Internal function.  Register a new MMC card with the driver model.
 */
int mmc_register_card(struct mmc_card *card)
{
    snprintf(card->dev.bus_id, sizeof(card->dev.bus_id),
        "%s:%04x", mmc_hostname(card->host), card->rca);

    /*
     * kobject를 object 계층구조에 add하고(즉, kset에 add하고), hotplug event를 발생시킨다.
     */
    return device_add(&card->dev);
}

- core.c
```

```
/**
 *     device_add - add device to device hierarchy.
 *     @dev:   device.
 *
 *     This is part 2 of device_register(), though may be called
 *     separately _iff_ device_initialize() has been called separately.
 *
 *     This adds it to the kobject hierarchy via kobject_add(), adds it
 *     to the global and sibling lists for the device, then
 *     adds it to the other relevant subsystems of the driver model.
 */
int device_add(struct device *dev)
{
       struct device *parent = NULL;
       int error = -EINVAL;

       dev = get_device(dev);
       if (!dev || !strlen(dev->bus_id))
              goto Error;

       parent = get_device(dev->parent);

       pr_debug("DEV: registering device: ID = '%s'\n", dev->bus_id);

       /* first, register with generic layer. */
       kobject_set_name(&dev->kobj, "%s", dev->bus_id);
       if (parent)
              dev->kobj.parent = &parent->kobj;

       if ((error = kobject_add(&dev->kobj)))
              goto Error;

       /*
        * 여기서 hotplug event가 발생된다.
        * 현재 이 device의 kset으로 devices_subsystem의 kset이
        * 할당되어 있기 때문에 밑에서 볼 kobject_hotplug()에서
        * 해당 kset의 hotplug_ops를 동작시킨다.

        * 이 hotplug_ops의 경우는, devices_subsystem declare시에
        * 등록이 된 dev_hotplug()이 실행이 되고, 이 녀석은
        * dev->bus->hotplug를 실행시킨다.
        * 즉, mmc의 경우 mmc_bus_type의 mmc_bus_hotplug()이 실행된다.
        */

       kobject_hotplug(&dev->kobj, KOBJ_ADD);

       if ((error = device_pm_add(dev)))
              goto PMError;
       if ((error = bus_add_device(dev)))
              goto BusError;
       if (parent)
              klist_add_tail(&parent->klist_children, &dev->knode_parent);

       /* notify platform of device entry */
       if (platform_notify)
              platform_notify(dev);
 Done:
       put_device(dev);
       return error;
 BusError:
       device_pm_remove(dev);
 PMError:
       kobject_hotplug(&dev->kobj, KOBJ_REMOVE);
       kobject_del(&dev->kobj);
 Error:
       if (parent)
              put_device(parent);
       goto Done;
}
```

- lib/kobject_uevent.c

```
#ifdef CONFIG_HOTPLUG
char hotplug_path[HOTPLUG_PATH_LEN] = "/sbin/hotplug";
u64 hotplug_seqnum;
static DEFINE_SPINLOCK(sequence_lock);

/**
 * kobject_hotplug - notify userspace by executing /sbin/hotplug
 *
 * @action: action that is happening (usually "ADD" or "REMOVE")
 * @kobj: struct kobject that the action is happening to
 */
void kobject_hotplug(struct kobject *kobj, enum kobject_action action)
{
    char *argv [3];
    char **envp = NULL;
    char *buffer = NULL;
    char *seq_buff;
    char *scratch;
    int i = 0;
    int retval;
    char *kobj_path = NULL;
    const char *name = NULL;
    char *action_string;
    u64 seq;
    struct kobject *top_kobj = kobj;
    struct kset *kset;
    static struct kset_hotplug_ops null_hotplug_ops;
    struct kset_hotplug_ops *hotplug_ops = &null_hotplug_ops;

    /* If this kobj does not belong to a kset,
       try to find a parent that does. */
    if (!top_kobj->kset && top_kobj->parent) {
       do {
             top_kobj = top_kobj->parent;
       } while (!top_kobj->kset && top_kobj->parent);
    }

    /*
     * mmc device의 경우 이 kset은 device_initiaize시 할당이 된
     * device_subsystem의 kset이 된다.
     */
    if (top_kobj->kset)
       kset = top_kobj->kset; // 당연히 할당되었으므로 있어야지롱.
    else
       return;

    /*
     * mmc의 경우 devices_subsystem으로 device가 붙게 되므로, device_hotplug_ops이
     * hotplug operation이 되고,
     */
    if (kset->hotplug_ops)
       hotplug_ops = kset->hotplug_ops;

    /* If the kset has a filter operation, call it.
       Skip the event, if the filter returns zero. */
    if (hotplug_ops->filter) {
       if (!hotplug_ops->filter(kset, kobj))
             return;
    }

    pr_debug ("%s\n", __FUNCTION__);

    action_string = action_to_string(action);
    if (!action_string)
       return;
```

```
envp = kmalloc(NUM_ENVP * sizeof (char *), GFP_KERNEL);
if (!envp)
   return;
memset (envp, 0x00, NUM_ENVP * sizeof (char *));

buffer = kmalloc(BUFFER_SIZE, GFP_KERNEL);
if (!buffer)
   goto exit;

if (hotplug_ops->name)
   name = hotplug_ops->name(kset, kobj);
if (name == NULL)
   name = kobject_name(&kset->kobj);

argv [0] = hotplug_path;
argv [1] = (char *)name; /* won't be changed but 'const' has to go */
argv [2] = NULL;

/* minimal command environment */
envp [i++] = "HOME=/";
envp [i++] = "PATH=/sbin:/bin:/usr/sbin:/usr/bin";

scratch = buffer;

envp [i++] = scratch;
scratch += sprintf(scratch, "ACTION=%s", action_string) + 1;

kobj_path = kobject_get_path(kobj, GFP_KERNEL);
if (!kobj_path)
   goto exit;

envp [i++] = scratch;
scratch += sprintf (scratch, "DEVPATH=%s", kobj_path) + 1;

envp [i++] = scratch;
scratch += sprintf(scratch, "SUBSYSTEM=%s", name) + 1;

/* reserve space for the sequence,
 * put the real one in after the hotplug call */
envp[i++] = seq_buff = scratch;
scratch += strlen("SEQNUM=18446744073709551616") + 1;

if (hotplug_ops->hotplug) {
   /* have the kset specific function add its stuff */


    /*
     * mmc의 경우 dev_hotplug()함수가 호출되고, 결국은 dev->bus->hotplug인
     * mmc_bus_hotplug()가 실행된다.
     */

   retval = hotplug_ops->hotplug (kset, kobj,
                &envp[i], NUM_ENVP - i, scratch,
                BUFFER_SIZE - (scratch - buffer));
   if (retval) {
         pr_debug ("%s - hotplug() returned %d\n",
                __FUNCTION__, retval);
         goto exit;
   }
}

spin_lock(&sequence_lock);
seq = ++hotplug_seqnum;
spin_unlock(&sequence_lock);
sprintf(seq_buff, "SEQNUM=%llu", (unsigned long long)seq);

pr_debug ("%s: %s %s seq=%llu %s %s %s %s %s\n",
     __FUNCTION__, argv[0], argv[1], (unsigned long long)seq,
     envp[0], envp[1], envp[2], envp[3], envp[4]);
```

```
    send_uevent(action_string, kobj_path, envp, GFP_KERNEL);

    if (!hotplug_path[0])
       goto exit;

    /*
     *  /sbin/hotplug가 실행된다.
     */
    retval = call_usermodehelper (argv[0], argv, envp, 0);
    if (retval)
       pr_debug ("%s - call_usermodehelper returned %d\n",
               __FUNCTION__, retval);

exit:
    kfree(kobj_path);
    kfree(buffer);
    kfree(envp);
    return;
}
EXPORT_SYMBOL(kobject_hotplug);
```

11.5. MMC/SD의 class interface

mmc/sd의 mmc_sysfs.c를 보면 mmc/sd의 경우 class interface도 사용을 한다. class의 경우도 hotplug를 사용하도록 되어 있다. class_device_initialize를 보면 class_obj_subsys에 등록이 되기 때문이다.

class관련 코드를 보면 class의 subsystem declare쪽을 보면 hotplug쪽이 선언되어 있지 않고, class_obj subsystem으로 넘어간 것을 볼 수 있다.

class_subsystem이 쓰이는 경우는 class_register등에서 사용이 되고, 해당 class에 platform device로
선언이 된 device를 class_device_initialize()이용해서 붙이는 경우는 class_obj_subsystem을 사용하게
됨으로써 hotplug처리가 된다.(해당 class에 대한)

```
- s3c2410mci.c or s3c2440mci.c

static int s3c2410sdi_probe(struct device *dev)
{
       struct platform_device  *pdev = to_platform_device(dev);
       struct mmc_host        *mmc;
       struct s3c2410sdi_host  *host;

       int ret;

       /*
        * platform device로 되어 있는 device를 class로 등록한다.
        */
       mmc = mmc_alloc_host(sizeof(struct s3c2410sdi_host), dev);
...

}

- mmc.c

/**
 *  mmc_alloc_host - initialise the per-host structure.
 *  @extra: sizeof private data structure
```

```
 *  @dev: pointer to host device model structure
 *
 *  Initialise the per-host structure.
 */
struct mmc_host *mmc_alloc_host(int extra, struct device *dev)
{
    struct mmc_host *host;

    host = mmc_alloc_host_sysfs(extra, dev); // platform device를 class의 device로 등록한다.
    if (host) {
       spin_lock_init(&host->lock);
       init_waitqueue_head(&host->wq);
       INIT_LIST_HEAD(&host->cards);
       INIT_WORK(&host->detect, mmc_rescan, host);

       /*
        * By default, hosts do not support SGIO or large requests.
        * They have to set these according to their abilities.
        */
       host->max_hw_segs = 1;
       host->max_phys_segs = 1;
       host->max_sectors = 1 << (PAGE_CACHE_SHIFT - 9);
       host->max_seg_size = PAGE_CACHE_SIZE;
    }

    return host;
}
```

- mmc_sysfs.c

```
/*
 * Internal function. Allocate a new MMC host.
 */
struct mmc_host *mmc_alloc_host_sysfs(int extra, struct device *dev)
{
    struct mmc_host *host;

    host = kmalloc(sizeof(struct mmc_host) + extra, GFP_KERNEL);
    if (host) {
       memset(host, 0, sizeof(struct mmc_host) + extra);

       host->dev = dev;
       host->class_dev.dev = host->dev; // 입력된 platform device인 s3c2410mci device를
class_device로 세팅하고

       host->class_dev.class = &mmc_host_class; // 해당 class_device의 class로 mmc_host_class로 등
록한다.
       class_device_initialize(&host->class_dev);
    }

    return host;
}
```

이 파일의 윗쪽을 보면 class가 이렇게 선언이 되어 잇다.

```
static struct class mmc_host_class = {
    .name           = "mmc_host",
    .release = mmc_host_classdev_release,
};
```

이 선언을 보면 hotplug는 빠져 잇는 것을 알 수 있다. 즉, 필요한 것중, 이름이랑 release함수만 선언했다.

- linux/device.h

```
/*
 * device classes
 */
struct class {
	const char		* name;
	struct module		* owner;

	struct subsystem	subsys;
	struct list_head	children;
	struct list_head	interfaces;
	struct semaphore	sem;	/* locks both the children and interfaces lists */

	struct class_attribute		* class_attrs;
	struct class_device_attribute	* class_dev_attrs;

	int	(*hotplug)(struct class_device *dev, char **envp,
			   int num_envp, char *buffer, int buffer_size);

	void	(*release)(struct class_device *dev);
	void	(*class_release)(struct class *class);
};


struct class_device {
	struct list_head	node;

	struct kobject		kobj;
	struct class		* class;	/* required */
	dev_t			devt;		/* dev_t, creates the sysfs "dev" */
	struct class_device_attribute *devt_attr;
	struct device		* dev;		/* not necessary, but nice to have */
	void			* class_data;	/* class-specific data */

	char	class_id[BUS_ID_SIZE];	/* unique to this class */
};
```

그리고, class_device는 다음과 같이 선언된다.

위의 mmc_alloc_host_sysfs()를 보면

```
...
	host->class_dev.class = &mmc_host_class; // 해당 class_device의 class로 mmc_host_class로 등록한다.
	class_device_initialize(&host->class_dev);
...
```

이렇게 선언이 된 mmc_host_class를 class_dev의 class로 할당을 한다. 여기서 보면 class에 hotplug가선언되어 있지 않으므로 class_device_initialize()에서 class_obj_subsystem으로 등록이 되어서class_obj_subsystem의 hotplug가 발생하더라도(class_device_add()시 kobject_hotplug()가 발생한다)

--> code를 보여주고 다시 설명

```
- class.c
이 함수는 mmc_alloc_host_sysfs()에서 호출됨
void class_device_initialize(struct class_device *class_dev)
{
	kobj_set_kset_s(class_dev, class_obj_subsys); // 여기서 kset이 설정되는데 class_obj_subsys로 되는구만, 일단 hotplug가 동작되도록 설계된 subsystem이다.
	kobject_init(&class_dev->kobj);
	INIT_LIST_HEAD(&class_dev->node);
}
```

- mmc_sysfs.c

```
/*
 * Internal function. Register a new MMC host with the MMC class.
 */
int mmc_add_host_sysfs(struct mmc_host *host)
{
    int err;

    if (!idr_pre_get(&mmc_host_idr, GFP_KERNEL))
       return -ENOMEM;

    spin_lock(&mmc_host_lock);
    err = idr_get_new(&mmc_host_idr, host, &host->index);
    spin_unlock(&mmc_host_lock);
    if (err)
       return err;

    snprintf(host->class_dev.class_id, BUS_ID_SIZE,
        "mmc%d", host->index);

    return class_device_add(&host->class_dev); // class device를 class subsystem에 등록한다.
}
```

- class.c

```
int class_device_add(struct class_device *class_dev)
{
       struct class * parent = NULL;
       struct class_interface * class_intf;
       int error;

       class_dev = class_device_get(class_dev);
       if (!class_dev)
              return -EINVAL;

       if (!strlen(class_dev->class_id)) {
              error = -EINVAL;
              goto register_done;
       }

       parent = class_get(class_dev->class);

       pr_debug("CLASS: registering class device: ID = '%s'\n",
              class_dev->class_id);

       /* first, register with generic layer. */
       kobject_set_name(&class_dev->kobj, "%s", class_dev->class_id);
       if (parent)
              class_dev->kobj.parent = &parent->subsys.kset.kobj;

       if ((error = kobject_add(&class_dev->kobj))) // kobject 계층구조에 붙이고,
              goto register_done;

       /* add the needed attributes to this device */
       if (MAJOR(class_dev->devt)) {
              struct class_device_attribute *attr;
              attr = kmalloc(sizeof(*attr), GFP_KERNEL);
              if (!attr) {
                     error = -ENOMEM;
                     kobject_del(&class_dev->kobj);
                     goto register_done;
              }
              memset(attr, sizeof(*attr), 0x00);
              attr->attr.name = "dev";
              attr->attr.mode = S_IRUGO;
              attr->attr.owner = parent->owner;
              attr->show = show_dev;
              attr->store = NULL;
```

```
                class_device_create_file(class_dev, attr);
                class_dev->devt_attr = attr;
        }

        class_device_add_attrs(class_dev);
        if (class_dev->dev)
                sysfs_create_link(&class_dev->kobj,
                                  &class_dev->dev->kobj, "device");

        /* notify any interfaces this device is now here */
        if (parent) {
                down(&parent->sem);
                list_add_tail(&class_dev->node, &parent->children);
                list_for_each_entry(class_intf, &parent->interfaces, node)
                        if (class_intf->add)
                                class_intf->add(class_dev);
                up(&parent->sem);
        }
        kobject_hotplug(&class_dev->kobj, KOBJ_ADD);  // add가 되었다는 hotplug()를 실행한다.

 register_done:
        if (error && parent)
                class_put(parent);
        class_device_put(class_dev);
        return error;
}
```

이 hotplug의 실행은 위의 mmc card device가 붙었을때와 상황은 같다. 하지만 여기서는 kset이 class_obj_subsystem의 kset이므로(class_device_initialize()를 볼 것) class_hotplug()가 호출된다.

```
static int class_hotplug(struct kset *kset, struct kobject *kobj, char **envp,
                         int num_envp, char *buffer, int buffer_size)
{
        struct class_device *class_dev = to_class_dev(kobj);
        int i = 0;
        int length = 0;
        int retval = 0;
...


       /* terminate, set to next free slot, shrink available space */
        envp[i] = NULL;
        envp = &envp[i];
        num_envp -= i;
        buffer = &buffer[length];
        buffer_size -= length;

        /*
         * 만일 class device의 class 멤버에 hotplug가 잇을 경우는 호출한다.
         * 여기서는 mmc의 경우이므로
         * class는

         * static struct class mmc_host_class = {
         *     .name        = "mmc_host",
         *     .release     = mmc_host_classdev_release,
         * };

         * 이다.
         *
         * 하지만 hotplug()함수는 설정이 안되어 있기 때문에
         * mmc host device(s3c2410mci device 자체, 즉, S3C2410의 platform device이다)
         * 가 등록이 되어있는 class_obj에서는 hotplug가 발생하지 않는다.
         */

        if (class_dev->class->hotplug) {
                /* have the bus specific function add its stuff */
                retval = class_dev->class->hotplug (class_dev, envp, num_envp,
                                                    buffer, buffer_size);
                        if (retval) {
```

```
                pr_debug ("%s - hotplug() returned %d\n",
                        __FUNCTION__, retval);
            }
        }

        return retval;
}
```

※ 참고로, 여기서 s3c24x0의 mmc host device는 platform device로 platform bus에 붙어 있다. 하지만 platform bus는 hotplug가 없기 때문에(bus_subsys를 볼 것) platform device는 hotplug가 발생하지 않는다(device subsystem에도 해당 device는 붙어 있지만 실제 device subsystem의 hotplug가 호출하는 것은 device_subsystem의 bus의 hotplug함수이다.).

하지만 해당 device가 hotplug를 지원하는 bus에 붙어 있다면(여기서와 같이 mmc bus) hotplug가 동작하게 된다.

※ 그리고, 여기서 설명하지 않은 block device의 hotplug는 card에 대한 hotplug처리처럼 mmc_block.c에서 register_blkdev()를 호출하면 drivers/block/genhd.c의 block_subsys에서 처리한다.

12. user-space에서의 hotplug처리

이 부분은 O'Reilly 사에서 출판된 Linux Device Driver 3rd Edition, chapter 14를 참조하기 바란다.

13. sysfs와 udev

sysfs에 대해서는 kernel 2.6.13 소스의 Documentation/filesystems/sysfs.txt 를 참조하고, udev에 대해서는 http://www.kroah.com/linux/talks/ols_2003_udev_talk/ 를 참조하기 바란다.

이 부분은 그냥 말로 해도 될 정도...^^

14. firmware upgrade

kernel 2.6.13 소스의 Documentation/firmware_class 디렉토리를 보면 자세한 예제와 howto가 잘 설명이 되어 있다.

15. aESOP-2440a에 대한 실제 log 분석

15.1. boot log에서의 device driver

```
U-Boot 1.1.2 (Oct 26 2005 - 20:59:41)

U-Boot code: 33C00000 -> 33C3FC78  BSS: -> 33C79518
RAM Configuration:
Bank #0: 30000000 64 MB
Get flash bank 0 size @ 0x0
Total Flash bank's sizes: 0x200000
```

```
protect monitor 3fc78 bytes @ address 0
Flash:  2 MB
NAND:64 MB
In:    serial
Out:   serial
Err:   serial

SD Initialize fail..

Hit any key to stop autoboot:  0
TFTP from server 172.16.1.200; our IP address is 172.16.1.100
Filename 'aesopk'.
Load address: 0x32000000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #####
done
Bytes transferred = 1352881 (14a4b1 hex)
## Booting image at 32000000 ...
   Image Name:   Linux-2.6.13-h1940-aesop2440
   Created:      2006-01-02   6:50:29 UTC
   Image Type:   ARM Linux Kernel Image (gzip compressed)
   Data Size:    1352817 Bytes =  1.3 MB
   Load Address: 30008000
   Entry Point:  30008000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK

Starting kernel ...

Linux version 2.6.13-h1940-aesop2440 (root@localhost.localdomain) (gcc version 3.3.4) #53 Mon
Jan 2 15:50:26 KST 2006
CPU: ARM920Tid(wb) [41129200] revision 0 (ARMv4T)
Machine: aESOP-2440
Memory policy: ECC disabled, Data cache writeback
CPU S3C2440A (id 0x32440001)
S3C2440: core 399.651 MHz, memory 133.217 MHz, peripheral 66.608 MHz
S3C2410 Clocks, (c) 2004 Simtec Electronics
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists
Kernel command line: root=/dev/nfs rw nfsroot=172.16.1.200:/korea-dokdo/nfsmount/2613 mem=63M
ip=172.16.1.100:172.16.1.200:172.16.1.1:255.255.255.0::eth0:off console=ttySAC0,115200n81
ethaddr=08:00:3e:26:0a:5b
irq: clearing subpending status 00000002
PID hash table entries: 256 (order: 8, 4096 bytes)
timer tcon=00500000, tcnt d8d2, tcfg 00000200,00000000, usec 0000170f
Console: colour dummy device 80x30
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 63MB = 63MB total
Memory: 60928KB available (2249K code, 581K data, 116K init)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok

// do_basic_setup()함수에서 driver_init() 호출, platform device와 bus등록

 9.PLATFORM: platform_bus_init() device_register call
 1.CORE    : DEV: registering device: ID = 'platform'
 9.PLATFORM: platform_bus_init() bus_register call
 8.BUS     : bus_register() bus type 'platform' registered
NET: Registered protocol family 16

// class subsystem에 device들 등록
 4.CLASS   : device class 'lcd': registering
 4.CLASS   : device class 'backlight': registering
 4.CLASS   : device class 'tty': registering
```

```
S3C2440: Initialising architecture

// aesop-2440 architecture specific device들을 platform bus에 등록
// 초기엔 주로 device들 add만 한다. driver는 뒷부분에 등록이 일어난다.
 9.PLATFORM: Registering platform device 's3c2440-uart.0'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2440-uart.0'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2440-uart.0
 8.BUS     : bus_add_device() bus: platform, add device: s3c2440-uart.0
 9.PLATFORM: Registering platform device 's3c2440-uart.1'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2440-uart.1'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2440-uart.1
 8.BUS     : bus_add_device() bus: platform, add device: s3c2440-uart.1
 9.PLATFORM: Registering platform device 's3c2440-uart.2'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2440-uart.2'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2440-uart.2
 8.BUS     : bus_add_device() bus: platform, add device: s3c2440-uart.2
 9.PLATFORM: Registering platform device 's3c2410-ohci'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-ohci'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-ohci
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-ohci
 9.PLATFORM: Registering platform device 's3c2410-lcd'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-lcd'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-lcd
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-lcd
 9.PLATFORM: Registering platform device 's3c2410-bl'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-bl'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-bl
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-bl
 9.PLATFORM: Registering platform device 's3c2410-wdt'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-wdt'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-wdt
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-wdt
 9.PLATFORM: Registering platform device 's3c2440-i2c'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2440-i2c'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2440-i2c
 8.BUS     : bus_add_device() bus: platform, add device: s3c2440-i2c
 9.PLATFORM: Registering platform device 's3c2410-iis'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-iis'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-iis
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-iis
 9.PLATFORM: Registering platform device 's3c2410-sdi'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-sdi'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-sdi
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-sdi
 9.PLATFORM: Registering platform device 's3c2410-usbgadget'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-usbgadget'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-usbgadget
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-usbgadget
 9.PLATFORM: Registering platform device 's3c2410-ts'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-ts'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-ts
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-ts
 9.PLATFORM: Registering platform device 's3c2440-nand'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2440-nand'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2440-nand
 8.BUS     : bus_add_device() bus: platform, add device: s3c2440-nand
 9.PLATFORM: Registering platform device 's3c2440-sound'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2440-sound'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2440-sound
 8.BUS     : bus_add_device() bus: platform, add device: s3c2440-sound
 9.PLATFORM: Registering platform device 's3c2410-buttons'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-buttons'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-buttons
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-buttons
 9.PLATFORM: Registering platform device 's3c2410-gd'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-gd'
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-gd
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-gd
 9.PLATFORM: Registering platform device 's3c2410-rtc'. Parent at platform
 1.CORE    : DEV: registering device: ID = 's3c2410-rtc'
```

```
 1.CORE    : dev_hotplug: dev->bus_id: s3c2410-rtc
 8.BUS     : bus_add_device() bus: platform, add device: s3c2410-rtc
S3C2440: IRQ Support
S3C2440: Clock Support, UPLL 47.980 MHz
 4.CLASS   : device class 'graphics': registering
 4.CLASS   : device class 'misc': registering
 4.CLASS   : device class 'scsi_host': registering
 8.BUS     : bus_register() bus type 'scsi' registered
 4.CLASS   : device class 'scsi_device': registering
SCSI subsystem initialized

// usb bus 등록
 8.BUS     : bus_register() bus type 'usb' registered
 4.CLASS   : device class 'usb_host': registering
 4.CLASS   : device class 'usb': registering
 8.BUS     : bus_add_driver() bus: usb, add driver: hub
usbcore: registered new driver hub
 8.BUS     : bus_add_driver() bus: usb, add driver: usb

// input class등록
 4.CLASS   : device class 'input': registering

 // i2c bus 등록
 8.BUS     : bus_register() bus type 'i2c' registered
 8.BUS     : bus_add_driver() bus: i2c, add driver: i2c_adapter
 4.CLASS   : device class 'i2c-adapter': registering
 4.CLASS   : device class 'net': registering
 4.CLASS   : device class 'mem': registering
 4.CLASS   : CLASS: registering class device: ID = 'mem'
 4.CLASS   : class_hotplug - name = mem
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'kmem'
 4.CLASS   : class_hotplug - name = kmem
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'null'
 4.CLASS   : class_hotplug - name = null
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'port'
 4.CLASS   : class_hotplug - name = port
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'zero'
 4.CLASS   : class_hotplug - name = zero
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'full'
 4.CLASS   : class_hotplug - name = full
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'random'
 4.CLASS   : class_hotplug - name = random
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'urandom'
 4.CLASS   : class_hotplug - name = urandom
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'kmsg'
 4.CLASS   : class_hotplug - name = kmsg
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'apm_bios'
 4.CLASS   : class_hotplug - name = apm_bios
 4.CLASS   : class_hotplug - hotplug run pre
S3C2410 DMA Driver, (c) 2003-2004 Simtec Electronics
DMA channel 0 at c4000000, irq 33
DMA channel 1 at c4000040, irq 34
DMA channel 2 at c4000080, irq 35
DMA channel 3 at c40000c0, irq 36
NetWinder Floating Point Emulator V0.97 (double precision)
yaffs Jan  2 2006 11:13:31 Installing.
Initializing Cryptographic API

// driver들을 bus에 등록
 8.BUS     : bus_add_driver() bus: platform, add driver: s3c2410-bl
```

```
// 현재 등록되는 driver와 매칭되는 device를 찾음
 3.DD     : platform: Matched Device s3c2410-bl with Driver s3c2410-bl
 4.CLASS  : CLASS: registering class device: ID = 's3c2410-bl'
 4.CLASS  : class_hotplug - name = s3c2410-bl
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 's3c2410-lcd'
 4.CLASS  : class_hotplug - name = s3c2410-lcd
 4.CLASS  : class_hotplug - hotplug run pre
s3c2410 Backlight Driver Initialized.

// 매칭된 device와 driver를 bind한다.
 3.DD     : bound device 's3c2410-bl' to driver 's3c2410-bl'
 3.DD     : platform: Bound Device s3c2410-bl to Driver s3c2410-bl
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-lcd
 3.DD     : platform: Matched Device s3c2410-lcd with Driver s3c2410-lcd
 4.CLASS  : CLASS: registering class device: ID = 'fb0'
 4.CLASS  : class_hotplug - name = fb0
 4.CLASS  : class_hotplug - hotplug run pre
Console: switching to colour frame buffer device 96x34
S3C24X0 fb0: s3c2410fb frame buffer device initialize done
 3.DD     : bound device 's3c2410-lcd' to driver 's3c2410-lcd'
 3.DD     : platform: Bound Device s3c2410-lcd to Driver s3c2410-lcd

// L3 bus는 godori가 platform bus로 안만들어 놨음....고칠까 말까?.....ㅋㅋ
GPIO L3 bus interface for S3C2440, installed

// tty class들이 주루룩 등록됨
 4.CLASS  : CLASS: registering class device: ID = 'tty'
 4.CLASS  : class_hotplug - name = tty
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'console'
 4.CLASS  : class_hotplug - name = console
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'ptmx'
 4.CLASS  : class_hotplug - name = ptmx
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'tty0'
 4.CLASS  : class_hotplug - name = tty0
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : device class 'vc': registering
 4.CLASS  : CLASS: registering class device: ID = 'vcs'
 4.CLASS  : class_hotplug - name = vcs
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'vcsa'
 4.CLASS  : class_hotplug - name = vcsa
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'tty1'
 4.CLASS  : class_hotplug - name = tty1
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'tty2'
 4.CLASS  : class_hotplug - name = tty2
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'tty3'
 4.CLASS  : class_hotplug - name = tty3
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'tty4'
 4.CLASS  : class_hotplug - name = tty4
 4.CLASS  : class_hotplug - hotplug run pre

....중략....

// RTC 등록
S3C2410 RTC, (c) 2004 Simtec Electronics
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-rtc
 3.DD     : platform: Matched Device s3c2410-rtc with Driver s3c2410-rtc
s3c2410-rtc s3c2410-rtc: rtc disabled, re-enabling
 4.CLASS  : CLASS: registering class device: ID = 'rtc'
 4.CLASS  : class_hotplug - name = rtc
 4.CLASS  : class_hotplug - hotplug run pre
 3.DD     : bound device 's3c2410-rtc' to driver 's3c2410-rtc'
```

```
 3.DD     : platform: Bound Device s3c2410-rtc to Driver s3c2410-rtc
 8.BUS    : bus_register() bus type 'serio' registered
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2440-uart
 3.DD     : platform: Matched Device s3c2440-uart.0 with Driver s3c2440-uart

// serial driver 등록
s3c2410_serial0 at MMIO 0x50000000 (irq = 70) is a S3C2440
 4.CLASS   : CLASS: registering class device: ID = 's3c2410_serial0'
 4.CLASS   : class_hotplug - name = s3c2410_serial0
 4.CLASS   : class_hotplug - hotplug run pre
 3.DD     : bound device 's3c2440-uart.0' to driver 's3c2440-uart'
 3.DD     : platform: Bound Device s3c2440-uart.0 to Driver s3c2440-uart
 3.DD     : platform: Matched Device s3c2440-uart.1 with Driver s3c2440-uart
s3c2410_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2440
 4.CLASS   : CLASS: registering class device: ID = 's3c2410_serial1'
 4.CLASS   : class_hotplug - name = s3c2410_serial1
 4.CLASS   : class_hotplug - hotplug run pre
 3.DD     : bound device 's3c2440-uart.1' to driver 's3c2440-uart'
 3.DD     : platform: Bound Device s3c2440-uart.1 to Driver s3c2440-uart
 3.DD     : platform: Matched Device s3c2440-uart.2 with Driver s3c2440-uart
s3c2410_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2440
 4.CLASS   : CLASS: registering class device: ID = 's3c2410_serial2'
 4.CLASS   : class_hotplug - name = s3c2410_serial2
 4.CLASS   : class_hotplug - hotplug run pre
 3.DD     : bound device 's3c2440-uart.2' to driver 's3c2440-uart'
 3.DD     : platform: Bound Device s3c2440-uart.2 to Driver s3c2440-uart
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered

// block device 등록(ramdisk관련인 듯): block subsystem은 hotplug가 존재하는 subsystem이다.
10.GENHD   : block_hotplug: ktype = block, minor = 0
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 1
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 2
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 3
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 4
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 5
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 6
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 7
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 8
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 9
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 10
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 11
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 12
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 13
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 14
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 15
10.GENHD   : block_hotplug: physdev->bus->name pre
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize

// loopback block device등록
10.GENHD   : block_hotplug: ktype = block, minor = 0
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 1
```

```
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 2
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 3
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 4
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 5
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 6
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 7
10.GENHD   : block_hotplug: physdev->bus->name pre
loop: loaded (max 8 devices)
nbd: registered device at major 43
10.GENHD   : block_hotplug: ktype = block, minor = 0
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 1
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 2
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 3
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 4
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 5
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 6
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 7
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 8
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 9
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 10
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 11
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 12
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 13
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 14
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: ktype = block, minor = 15
10.GENHD   : block_hotplug: physdev->bus->name pre
 4.CLASS   : CLASS: registering class device: ID = 'lo'
 4.CLASS   : class_hotplug - name = lo
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : class_hotplug - hotplug ===> run

// cs8900은 kernel driver model hierarchy에 등록시키지 않았다. 어차피 단말에서는 빼버릴테니...
// 그냥 kernel 2.4.x의 driver형태로만 되어 있음.
cs89x0:cs89x0_probe(0x0)
PP_addr=0x3000
cs89x0.c: v2.4.3-pre1 Russell Nelson <nelson@crynwr.com>, Andrew Morton <andrewm@uow.edu.au>
eth0: cs8900 rev K found at 0xf8000300
cs89x0: Extended EEPROM checksum bad and no Cirrus EEPROM, relying on command line
cs89x0 media RJ-45, IRQ 53, programmed I/O, MAC 00:00:c0:ff:ee:08
cs89x0_probe1() successful

// cs8900이 network device로 등록이 되면서, class로는 등록이 된다.
 4.CLASS   : CLASS: registering class device: ID = 'eth0'
 4.CLASS   : class_hotplug - name = eth0
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : class_hotplug - hotplug ===> run
cs89x0:cs89x0_probe(0x0)
cs89x0: request_region(0xf8000300, 0x10) failed
cs89x0: no cs8900 or cs8920 detected.  Be sure to disable PnP with SETUP
```

```
 8.BUS    : bus_add_driver() bus: scsi, add driver: sd
 4.CLASS  : device class 'mtd': registering
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2440-nand
 3.DD     : platform: Matched Device s3c2440-nand with Driver s3c2440-nand
s3c2410-nand: mapped registers at c4080000
s3c2410-nand: timing: Tacls 13ns, Twrph0 66ns, Twrph1 39ns
NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MiB 3,3V 8-bit)
Scanning device for bad blocks
Bad eraseblock 1090 at 0x01108000
Creating 1 MTD partitions on "NAND 64MiB 3,3V 8-bit":
0x00000000-0x04000000 : "Total nand"
10.GENHD  : block_hotplug: ktype = block, minor = 0
10.GENHD  : block_hotplug: physdev->bus->name pre
 4.CLASS  : CLASS: registering class device: ID = 'mtd0'
 4.CLASS  : class_hotplug - name = mtd0
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'mtd0ro'
 4.CLASS  : class_hotplug - name = mtd0ro
 4.CLASS  : class_hotplug - hotplug run pre
 3.DD     : bound device 's3c2440-nand' to driver 's3c2440-nand'
 3.DD     : platform: Bound Device s3c2440-nand to Driver s3c2440-nand
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-nand
usbmon: debugfs is not available
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-ohci
 3.DD     : platform: Matched Device s3c2410-ohci with Driver s3c2410-ohci
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
 4.CLASS  : CLASS: registering class device: ID = 'usb1'
 4.CLASS  : class_hotplug - name = usb1
 4.CLASS  : class_hotplug - hotplug run pre
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number 1
s3c2410-ohci s3c2410-ohci: irq 42, io mem 0x49000000
 1.CORE   : DEV: registering device: ID = 'usb1'
 1.CORE   : dev_hotplug: dev->bus_id: usb1
 1.CORE   : dev_hotplug: hotplug --> call
 8.BUS    : bus_add_device() bus: usb, add device: usb1
 3.DD     : bound device 'usb1' to driver 'usb'
 1.CORE   : DEV: registering device: ID = '1-0:1.0'
 1.CORE   : dev_hotplug: dev->bus_id: 1-0:1.0
 1.CORE   : dev_hotplug: hotplug --> call
 8.BUS    : bus_add_device() bus: usb, add device: 1-0:1.0
 3.DD     : usb: Matched Device 1-0:1.0 with Driver hub
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
 3.DD     : bound device '1-0:1.0' to driver 'hub'
 3.DD     : usb: Bound Device 1-0:1.0 to Driver hub
 3.DD     : bound device 's3c2410-ohci' to driver 's3c2410-ohci'
 3.DD     : platform: Bound Device s3c2410-ohci to Driver s3c2410-ohci
Initializing USB Mass Storage driver...
 8.BUS    : bus_add_driver() bus: usb, add driver: usb-storage
usbcore: registered new driver usb-storage
USB Mass Storage support registered.
s3c2410_udc: version 28 Aug 2005
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-usbgadget
 3.DD     : platform: Matched Device s3c2410-usbgadget with Driver s3c2410-usbgadget
 3.DD     : bound device 's3c2410-usbgadget' to driver 's3c2410-usbgadget'
 3.DD     : platform: Bound Device s3c2410-usbgadget to Driver s3c2410-usbgadget
 4.CLASS  : CLASS: registering class device: ID = 'mice'
 4.CLASS  : class_hotplug - name = mice
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'psaux'
 4.CLASS  : class_hotplug - name = psaux
 4.CLASS  : class_hotplug - hotplug run pre
mice: PS/2 mouse device common for all mice
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-buttons
 3.DD     : platform: Matched Device s3c2410-buttons with Driver s3c2410-buttons
s3c2410-buttons successfully loaded
 4.CLASS  : CLASS: registering class device: ID = 'event0'
 4.CLASS  : class_hotplug - name = event0
 4.CLASS  : class_hotplug - hotplug run pre
```

```
 3.DD     : bound device 's3c2410-buttons' to driver 's3c2410-buttons'
 3.DD     : platform: Bound Device s3c2410-buttons to Driver s3c2410-buttons
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-gd
 3.DD     : platform: Matched Device s3c2410-gd with Driver s3c2410-gd
s3c2410-gd successfully loaded
 3.DD     : bound device 's3c2410-gd' to driver 's3c2410-gd'
 3.DD     : platform: Bound Device s3c2410-gd to Driver s3c2410-gd
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-ts
 3.DD     : platform: Matched Device s3c2410-ts with Driver s3c2410-ts
s3c2410 TouchScreen successfully loaded
 4.CLASS  : CLASS: registering class device: ID = 'mouse0'
 4.CLASS  : class_hotplug - name = mouse0
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'event1'
 4.CLASS  : class_hotplug - name = event1
 4.CLASS  : class_hotplug - hotplug run pre
 3.DD     : bound device 's3c2410-ts' to driver 's3c2410-ts'
 3.DD     : platform: Bound Device s3c2410-ts to Driver s3c2410-ts
i2c /dev entries driver
 4.CLASS  : device class 'i2c-dev': registering
 8.BUS    : bus_add_driver() bus: i2c, add driver: dev_driver
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-i2c
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2440-i2c
 3.DD     : platform: Matched Device s3c2440-i2c with Driver s3c2440-i2c
s3c2440-i2c s3c2440-i2c: slave address 0x10
s3c2440-i2c s3c2440-i2c: bus frequency set to 378 KHz
 1.CORE   : DEV: registering device: ID = 'i2c-0'
 4.CLASS  : CLASS: registering class device: ID = 'i2c-0'
 4.CLASS  : class_hotplug - name = i2c-0
 4.CLASS  : class_hotplug - hotplug run pre
 4.CLASS  : CLASS: registering class device: ID = 'i2c-0'
 4.CLASS  : class_hotplug - name = i2c-0
 4.CLASS  : class_hotplug - hotplug run pre
s3c2440-i2c s3c2440-i2c: i2c-0: S3C I2C adapter
 3.DD     : bound device 's3c2440-i2c' to driver 's3c2440-i2c'
 3.DD     : platform: Bound Device s3c2440-i2c to Driver s3c2440-i2c


// mmc/sd 관련 루틴

// bus와 class를 등록
 6.MMCSYSFS: mmc_init, bus_register/class_register() call
 8.BUS    : bus_register() bus type 'mmc' registered
 4.CLASS  : device class 'mmc_host': registering

// block device로 등록
 7.MMCBLOCK:  register_blkdev after

// block driver를 mmc bus의 driver로 등록
 6.MMCSYSFS: mmc_register_driver
 8.BUS    : bus_add_driver() bus: mmc, add driver: mmcblk

// platform bus에 s3c2410-sdi driver 등록
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2410-sdi

// 이미 platform device로 등록되어 있던 device인 "s3c2410-sdi"과 driver인 "s3c2410-sdi" match확인
 3.DD     : platform: Matched Device s3c2410-sdi with Driver s3c2410-sdi

// device와 driver가 매칭이 되었으니 driver->probe() 호출,
// probe가 호출되면 s3c2410-sdi platform device를 mmc class의 class device로 등록한다.
 8.2440MCI : s3c2410sdi_probe, mmc_alloc_host() call
 5.MMC    : mmc_alloc_host, mmc_alloc_host_sysfs() call
 6.MMCSYSFS: mmc_alloc_host_sysfs, class_device_initialize() call
mmci-s3c2410: probe: mapped sdi_base=c4680000 irq=37 irq_cd=62 dma=0.
 6.MMCSYSFS: mmc_add_host_sysfs, class_device_add() call
 4.CLASS  : CLASS: registering class device: ID = 'mmc0'
 4.CLASS  : class_hotplug - name = mmc0
 4.CLASS  : class_hotplug - hotplug run pre
mmci-s3c2410: initialisation done.
```

```
// device인 "s3c2410-sdi"과 driver인 "s3c2410-sdi" bine시킨다.
// sysfs의 platform bus 밑의 mmc device와 mmc driver에 대한 soft link를 건다.
 3.DD     : bound device 's3c2410-sdi' to driver 's3c2410-sdi'
 3.DD     : platform: Bound Device s3c2410-sdi to Driver s3c2410-sdi
 4.CLASS   : device class 'sound': registering
godori: AESOP2440 SOUND driver register
 8.BUS    : bus_add_driver() bus: platform, add driver: s3c2440-sound
 3.DD     : platform: Matched Device s3c2440-sound with Driver s3c2440-sound
godori: AESOP2440 SOUND driver.........probe
 4.CLASS   : CLASS: registering class device: ID = 'dsp'
 4.CLASS   : class_hotplug - name = dsp
 4.CLASS   : class_hotplug - hotplug run pre
 4.CLASS   : CLASS: registering class device: ID = 'mixer'
 4.CLASS   : class_hotplug - name = mixer
 4.CLASS   : class_hotplug - hotplug run pre
AESOP2440 UDA1341 audio driver initialized
 3.DD     : bound device 's3c2440-sound' to driver 's3c2440-sound'
 3.DD     : platform: Bound Device s3c2440-sound to Driver s3c2440-sound
NET: Registered protocol family 2
IP route cache hash table entries: 512 (order: -1, 2048 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
TCP bic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
eth0: using half-duplex 10Base-T (RJ-45)
MMC: sd_app_op_cond timed out. Probably no SD-Card here.
 5.MMC    : mmc_setup, mmc_discover_cards(SD) call
 5.MMC    : mmc_setup, mmc_discover_cards(MMC) call
IP-Config: Complete:
     device=eth0, addr=172.16.1.100, mask=255.255.255.0, gw=172.16.1.1,
    host=172.16.1.100, domain=, nis-domain=(none),
    bootserver=172.16.1.200, rootserver=172.16.1.200, rootpath=
Looking up port of RPC 100003/2 on 172.16.1.200
Looking up port of RPC 100005/1 on 172.16.1.200
VFS: Mounted root (nfs filesystem).
Freeing init memory: 116K
INIT: version 2.86 booting
Initializing udev dynamic device directory.
mount: Mounting /dev/root on / failed: No such file or directory
INIT: Entering runlevel: 3
[: 0: unknown operand
mount: Mounting /dev/mmcblk0p1 on /mnt/mmcblk0p1 failed: No such file or directory


Linux 2.6.13-h1940-aesop2440.

 /) /)
(='.'=)

login[290]: root login  on `tts/0'
```

15.2. 부팅후 sysfs tree

- 카드 삽입전의 sysfs에서의 tree

```
sys
|-- block
|   |-- loop0
|   |-- loop1
|   |-- loop2
|   |-- loop3
|   |-- loop4
|   |-- loop5
|   |-- loop6
```

```
|   |-- loop7
|   |-- mtdblock0
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd0
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd1
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd10
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd11
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd12
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd13
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd14
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd15
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd2
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd3
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd4
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd5
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd6
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd7
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd8
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd9
|   |   `-- queue
|   |       `-- iosched
|   |-- ram0
|   |-- ram1
|   |-- ram10
|   |-- ram11
|   |-- ram12
|   |-- ram13
|   |-- ram14
|   |-- ram15
|   |-- ram2
|   |-- ram3
|   |-- ram4
|   |-- ram5
|   |-- ram6
|   |-- ram7
|   |-- ram8
|   `-- ram9
|-- bus
|   |-- i2c
```

```
|   |   |-- devices
|   |   `-- drivers
|   |       |-- dev_driver
|   |       `-- i2c_adapter
|   |-- mmc
|   |   |-- devices
|   |   `-- drivers
|   |       `-- mmcblk
|   |-- platform
|   |   |-- devices
|   |   |   |-- s3c2410-bl -> ../../../devices/platform/s3c2410-bl
|   |   |   |-- s3c2410-buttons -> ../../../devices/platform/s3c2410-buttons
|   |   |   |-- s3c2410-iis -> ../../../devices/platform/s3c2410-iis
|   |   |   |-- s3c2410-lcd -> ../../../devices/platform/s3c2410-lcd
|   |   |   |-- s3c2410-ohci -> ../../../devices/platform/s3c2410-ohci
|   |   |   |-- s3c2410-rtc -> ../../../devices/platform/s3c2410-rtc
|   |   |   |-- s3c2410-sdi -> ../../../devices/platform/s3c2410-sdi
|   |   |   |-- s3c2410-ts -> ../../../devices/platform/s3c2410-ts
|   |   |   |-- s3c2410-usbgadget -> ../../../devices/platform/s3c2410-usbgadget
|   |   |   |-- s3c2410-wdt -> ../../../devices/platform/s3c2410-wdt
|   |   |   |-- s3c2440-i2c -> ../../../devices/platform/s3c2440-i2c
|   |   |   |-- s3c2440-nand -> ../../../devices/platform/s3c2440-nand
|   |   |   |-- s3c2440-sound -> ../../../devices/platform/s3c2440-sound
|   |   |   |-- s3c2440-uart.0 -> ../../../devices/platform/s3c2440-uart.0
|   |   |   |-- s3c2440-uart.1 -> ../../../devices/platform/s3c2440-uart.1
|   |   |   `-- s3c2440-uart.2 -> ../../../devices/platform/s3c2440-uart.2
|   |   `-- drivers
|   |       |-- s3c2410-bl
|   |       |   `-- s3c2410-bl -> ../../../../devices/platform/s3c2410-bl
|   |       |-- s3c2410-buttons
|   |       |   `-- s3c2410-buttons -> ../../../../devices/platform/s3c2410-buttons
|   |       |-- s3c2410-i2c
|   |       |-- s3c2410-lcd
|   |       |   `-- s3c2410-lcd -> ../../../../devices/platform/s3c2410-lcd
|   |       |-- s3c2410-nand
|   |       |-- s3c2410-ohci
|   |       |   `-- s3c2410-ohci -> ../../../../devices/platform/s3c2410-ohci
|   |       |-- s3c2410-rtc
|   |       |   `-- s3c2410-rtc -> ../../../../devices/platform/s3c2410-rtc
|   |       |-- s3c2410-sdi
|   |       |   `-- s3c2410-sdi -> ../../../../devices/platform/s3c2410-sdi
|   |       |-- s3c2410-ts
|   |       |   `-- s3c2410-ts -> ../../../../devices/platform/s3c2410-ts
|   |       |-- s3c2410-usbgadget
|   |       |   `-- s3c2410-usbgadget -> ../../../../devices/platform/s3c2410-usbgadget
|   |       |-- s3c2440-i2c
|   |       |   `-- s3c2440-i2c -> ../../../../devices/platform/s3c2440-i2c
|   |       |-- s3c2440-nand
|   |       |   `-- s3c2440-nand -> ../../../../devices/platform/s3c2440-nand
|   |       |-- s3c2440-sound
|   |       |   `-- s3c2440-sound -> ../../../../devices/platform/s3c2440-sound
|   |       `-- s3c2440-uart
|   |           |-- s3c2440-uart.0 -> ../../../../devices/platform/s3c2440-uart.0
|   |           |-- s3c2440-uart.1 -> ../../../../devices/platform/s3c2440-uart.1
|   |           `-- s3c2440-uart.2 -> ../../../../devices/platform/s3c2440-uart.2
|   |-- scsi
|   |   |-- devices
|   |   `-- drivers
|   |       `-- sd
|   |-- serio
|   |   |-- devices
|   |   `-- drivers
|   `-- usb
|       |-- devices
|       |   |-- 1-0:1.0 -> ../../../devices/platform/s3c2410-ohci/usb1/1-0:1.0
|       |   `-- usb1 -> ../../../devices/platform/s3c2410-ohci/usb1
|       `-- drivers
|           |-- hub
|           |   `-- 1-0:1.0 -> ../../../../devices/platform/s3c2410-ohci/usb1/1-0:1.0
|           |-- usb
```

```
|         |   `-- usb1 -> ../../../../devices/platform/s3c2410-ohci/usb1
|         `-- usb-storage
|-- class
|   |-- backlight
|   |   `-- s3c2410-bl
|   |-- graphics
|   |   `-- fb0
|   |-- i2c-adapter
|   |   `-- i2c-0
|   |       `-- device -> ../../../devices/platform/s3c2440-i2c/i2c-0
|   |-- i2c-dev
|   |   `-- i2c-0
|   |       `-- device -> ../../../devices/platform/s3c2440-i2c
|   |-- input
|   |   |-- event0
|   |   |-- event1
|   |   |-- mice
|   |   `-- mouse0
|   |-- lcd
|   |   `-- s3c2410-lcd
|   |-- mem
|   |   |-- full
|   |   |-- kmem
|   |   |-- kmsg
|   |   |-- mem
|   |   |-- null
|   |   |-- port
|   |   |-- random
|   |   |-- urandom
|   |   `-- zero
|   |-- misc
|   |   |-- apm_bios
|   |   |-- psaux
|   |   `-- rtc
|   |-- mmc_host
|   |   `-- mmc0
|   |       `-- device -> ../../../devices/platform/s3c2410-sdi
|   |-- mtd
|   |   |-- mtd0
|   |   `-- mtd0ro
|   |-- net
|   |   |-- eth0
|   |   |   `-- statistics
|   |   `-- lo
|   |       `-- statistics
|   |-- scsi_device
|   |-- scsi_host
|   |-- sound
|   |   |-- dsp
|   |   `-- mixer
|   |-- tty
|   |   |-- console
|   |   |-- ptmx
|   |   |-- s3c2410_serial0
|   |   |   `-- device -> ../../../devices/platform/s3c2440-uart.0
|   |   |-- s3c2410_serial1
|   |   |   `-- device -> ../../../devices/platform/s3c2440-uart.1
|   |   |-- s3c2410_serial2
|   |   |   `-- device -> ../../../devices/platform/s3c2440-uart.2
|   |   |-- tty
|   |   |-- tty0
|   |   |-- tty1
|   |   |-- tty10
|   |   |-- tty11
|   |   |-- tty12
|   |   |-- tty13
|   |   |-- tty14
|   |   |-- tty15
|   |   |-- tty16
|   |   |-- tty17
```

```
|   |   |-- tty18
|   |   |-- tty19
|   |   |-- tty2
|   |   |-- tty20
|   |   |-- tty21
|   |   |-- tty22
|   |   |-- tty23
|   |   |-- tty24
|   |   |-- tty25
|   |   |-- tty26
|   |   |-- tty27
|   |   |-- tty28
|   |   |-- tty29
|   |   |-- tty3
|   |   |-- tty30
|   |   |-- tty31
|   |   |-- tty32
|   |   |-- tty33
|   |   |-- tty34
|   |   |-- tty35
|   |   |-- tty36
|   |   |-- tty37
|   |   |-- tty38
|   |   |-- tty39
|   |   |-- tty4
|   |   |-- tty40
|   |   |-- tty41
|   |   |-- tty42
|   |   |-- tty43
|   |   |-- tty44
|   |   |-- tty45
|   |   |-- tty46
|   |   |-- tty47
|   |   |-- tty48
|   |   |-- tty49
|   |   |-- tty5
|   |   |-- tty50
|   |   |-- tty51
|   |   |-- tty52
|   |   |-- tty53
|   |   |-- tty54
|   |   |-- tty55
|   |   |-- tty56
|   |   |-- tty57
|   |   |-- tty58
|   |   |-- tty59
|   |   |-- tty6
|   |   |-- tty60
|   |   |-- tty61
|   |   |-- tty62
|   |   |-- tty63
|   |   |-- tty7
|   |   |-- tty8
|   |   `-- tty9
|   |-- usb
|   |-- usb_host
|   |   `-- usb1
|   |       `-- device -> ../../../devices/platform/s3c2410-ohci
|   `-- vc
|       |-- vcs
|       `-- vcsa
|-- devices
|   |-- platform
|   |   |-- power
|   |   |-- s3c2410-bl
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-bl
|   |   |   `-- power
|   |   |-- s3c2410-buttons
|   |   |   |-- bus -> ../../../bus/platform
```

```
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-buttons
|   |   |   `-- power
|   |   |-- s3c2410-iis
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   `-- power
|   |   |-- s3c2410-lcd
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-lcd
|   |   |   `-- power
|   |   |-- s3c2410-ohci
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-ohci
|   |   |   |-- power
|   |   |   `-- usb1
|   |   |      |-- 1-0:1.0
|   |   |      |   |-- bus -> ../../../../../bus/usb
|   |   |      |   |-- driver -> ../../../../../bus/usb/drivers/hub
|   |   |      |   `-- power
|   |   |      |-- bus -> ../../../../bus/usb
|   |   |      |-- driver -> ../../../../bus/usb/drivers/usb
|   |   |      `-- power
|   |   |-- s3c2410-rtc
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-rtc
|   |   |   `-- power
|   |   |-- s3c2410-sdi
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-sdi
|   |   |   `-- power
|   |   |-- s3c2410-ts
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-ts
|   |   |   `-- power
|   |   |-- s3c2410-usbgadget
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-usbgadget
|   |   |   `-- power
|   |   |-- s3c2410-wdt
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   `-- power
|   |   |-- s3c2440-i2c
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-i2c
|   |   |   |-- i2c-0
|   |   |   |   `-- power
|   |   |   `-- power
|   |   |-- s3c2440-nand
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-nand
|   |   |   `-- power
|   |   |-- s3c2440-sound
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-sound
|   |   |   `-- power
|   |   |-- s3c2440-uart.0
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-uart
|   |   |   `-- power
|   |   |-- s3c2440-uart.1
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-uart
|   |   |   `-- power
|   |   `-- s3c2440-uart.2
|   |      |-- bus -> ../../../bus/platform
|   |      |-- driver -> ../../../bus/platform/drivers/s3c2440-uart
|   |      `-- power
|   `-- system
|      |-- cpu
|      |   `-- cpu0
|      |-- s3c2440-core
```

```
|      |   `-- s3c2440-core0
|      |-- s3c24xx-dma
|      |   |-- s3c24xx-dma0
|      |   |-- s3c24xx-dma1
|      |   |-- s3c24xx-dma2
|      |   `-- s3c24xx-dma3
|      `-- timer
|          `-- timer0
|-- firmware
|-- kernel
|-- module
|   |-- lockd
|   |   `-- parameters
|   |-- mmc_block
|   |   `-- parameters
|   |-- nbd
|   |   `-- parameters
|   |-- scsi_mod
|   |   `-- parameters
|   |-- tcp_bic
|   |   `-- parameters
|   |-- usb_storage
|   |   `-- parameters
|   `-- usbcore
|       `-- parameters
`-- power
```

15.3. SD card 삽입에 따른 시스템 동작과 hotplug

sd카드 삽입에 따른 kernel단에서의 sd card hotplug

15.3.1. SD card 삽입 log 및 hotplug동작

```
root@godori:~# ghc: interrupt occur ***********************
ghc: interrupt occur ***********************
ghc: interrupt occur ***********************
ghc: interrupt occur ***********************
MMC: sd_app_op_cond: at least one card is busy - trying again.
MMC: sd_app_op_cond: at least one card is busy - trying again.
MMC: sd_app_op_cond: at least one card is busy - trying again.
MMC: sd_app_op_cond: at least one card is busy - trying again.
 5.MMC     : mmc_setup, mmc_discover_cards(SD) call
 5.MMC     : mmc_discover_cards, mmc_alloc_card(SD) call
 5.MMC     : mmc_alloc_card, mmc_init_card() call
 6.MMCSYSFS: mmc_init_card, device_initialize() call
 5.MMC     : mmc_setup, mmc_discover_cards(MMC) call
 5.MMC     : mmc_rescan, mmc_register_card() call
 6.MMCSYSFS: mmc_register_card, device_add() call

// 삽입된 mmc card를 device 등록
 1.CORE    : DEV: registering device: ID = 'mmc0:a95c'
 1.CORE    : dev_hotplug: dev->bus_id: mmc0:a95c

// 이때 해당 device가 속한 subsystem의 kset의 hotplug호출(devices_subsystem)되고,
// 해당 device가 속한 bus의 hotplug함수가 호출된다.
 1.CORE    : dev_hotplug: hotplug --> call
 6.MMCSYSFS: mmc_bus_hotplug run

// mmc bus에 device를 등록하고
 8.BUS     : bus_add_device() bus: mmc, add device: mmc0:a95c

// matching되는 block driver를 찾고
 3.DD      : mmc: Matched Device mmc0:a95c with Driver mmcblk
```

```
// block device를 찾았다는 hotplug 발생(block_subsystem에서)
mmcblk0: mmc0:a95c SD512 495488KiB
10.GENHD   : block_hotplug: ktype = block, minor = 0
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: physdev->bus->name = mmc

// block device에서 partition을 찾았다고 hotplug발생(block_subsystem에서)
 mmcblk0:<7>MMC: starting cmd 37 arg a95c0000 flags 00000009
 p1
10.GENHD   : block_hotplug: ktype = part , minor = 1
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: physdev->bus->name = mmc

// mmc bus에 있는 device와 mmc block driver를 bind한다.
 3.DD      : bound device 'mmc0:a95c' to driver 'mmcblk'
 3.DD      : mmc: Bound Device mmc0:a95c to Driver mmcblk
```

15.3.2. 카드 삽입후의 sysfs에서의 tree구조

```
sys
|-- block
|   |-- loop0
|   |-- loop1
|   |-- loop2
|   |-- loop3
|   |-- loop4
|   |-- loop5
|   |-- loop6
|   |-- loop7
|   |-- mmcblk0
|   |   |-- device -> ../../devices/platform/s3c2410-sdi/mmc0:a95c
|   |   |-- mmcblk0p1
|   |   `-- queue
|   |       `-- iosched
|   |-- mtdblock0
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd0
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd1
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd10
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd11
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd12
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd13
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd14
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd15
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd2
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd3
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd4
```

```
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd5
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd6
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd7
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd8
|   |   `-- queue
|   |       `-- iosched
|   |-- nbd9
|   |   `-- queue
|   |       `-- iosched
|   |-- ram0
|   |-- ram1
|   |-- ram10
|   |-- ram11
|   |-- ram12
|   |-- ram13
|   |-- ram14
|   |-- ram15
|   |-- ram2
|   |-- ram3
|   |-- ram4
|   |-- ram5
|   |-- ram6
|   |-- ram7
|   |-- ram8
|   `-- ram9
|-- bus
|   |-- i2c
|   |   |-- devices
|   |   `-- drivers
|   |       |-- dev_driver
|   |       `-- i2c_adapter
|   |-- mmc
|   |   |-- devices
|   |   |   `-- mmc0:a95c -> ../../../devices/platform/s3c2410-sdi/mmc0:a95c
|   |   `-- drivers
|   |       `-- mmcblk
|   |           `-- mmc0:a95c -> ../../../../devices/platform/s3c2410-sdi/mmc0:a95c
|   |-- platform
|   |   |-- devices
|   |   |   |-- s3c2410-bl -> ../../../devices/platform/s3c2410-bl
|   |   |   |-- s3c2410-buttons -> ../../../devices/platform/s3c2410-buttons
|   |   |   |-- s3c2410-iis -> ../../../devices/platform/s3c2410-iis
|   |   |   |-- s3c2410-lcd -> ../../../devices/platform/s3c2410-lcd
|   |   |   |-- s3c2410-ohci -> ../../../devices/platform/s3c2410-ohci
|   |   |   |-- s3c2410-rtc -> ../../../devices/platform/s3c2410-rtc
|   |   |   |-- s3c2410-sdi -> ../../../devices/platform/s3c2410-sdi
|   |   |   |-- s3c2410-ts -> ../../../devices/platform/s3c2410-ts
|   |   |   |-- s3c2410-usbgadget -> ../../../devices/platform/s3c2410-usbgadget
|   |   |   |-- s3c2410-wdt -> ../../../devices/platform/s3c2410-wdt
|   |   |   |-- s3c2440-i2c -> ../../../devices/platform/s3c2440-i2c
|   |   |   |-- s3c2440-nand -> ../../../devices/platform/s3c2440-nand
|   |   |   |-- s3c2440-sound -> ../../../devices/platform/s3c2440-sound
|   |   |   |-- s3c2440-uart.0 -> ../../../devices/platform/s3c2440-uart.0
|   |   |   |-- s3c2440-uart.1 -> ../../../devices/platform/s3c2440-uart.1
|   |   |   `-- s3c2440-uart.2 -> ../../../devices/platform/s3c2440-uart.2
|   |   `-- drivers
|   |       |-- s3c2410-bl
|   |       |   `-- s3c2410-bl -> ../../../../devices/platform/s3c2410-bl
|   |       |-- s3c2410-buttons
|   |       |   `-- s3c2410-buttons -> ../../../../devices/platform/s3c2410-buttons
|   |       |-- s3c2410-i2c
|   |       |-- s3c2410-lcd
```

```
|   |      |   `-- s3c2410-lcd -> ../../../../devices/platform/s3c2410-lcd
|   |      |-- s3c2410-nand
|   |      |-- s3c2410-ohci
|   |      |   `-- s3c2410-ohci -> ../../../../devices/platform/s3c2410-ohci
|   |      |-- s3c2410-rtc
|   |      |   `-- s3c2410-rtc -> ../../../../devices/platform/s3c2410-rtc
|   |      |-- s3c2410-sdi
|   |      |   `-- s3c2410-sdi -> ../../../../devices/platform/s3c2410-sdi
|   |      |-- s3c2410-ts
|   |      |   `-- s3c2410-ts -> ../../../../devices/platform/s3c2410-ts
|   |      |-- s3c2410-usbgadget
|   |      |   `-- s3c2410-usbgadget -> ../../../../devices/platform/s3c2410-usbgadget
|   |      |-- s3c2440-i2c
|   |      |   `-- s3c2440-i2c -> ../../../../devices/platform/s3c2440-i2c
|   |      |-- s3c2440-nand
|   |      |   `-- s3c2440-nand -> ../../../../devices/platform/s3c2440-nand
|   |      |-- s3c2440-sound
|   |      |   `-- s3c2440-sound -> ../../../../devices/platform/s3c2440-sound
|   |      `-- s3c2440-uart
|   |         |-- s3c2440-uart.0 -> ../../../../devices/platform/s3c2440-uart.0
|   |         |-- s3c2440-uart.1 -> ../../../../devices/platform/s3c2440-uart.1
|   |         `-- s3c2440-uart.2 -> ../../../../devices/platform/s3c2440-uart.2
|   |-- scsi
|   |   |-- devices
|   |   `-- drivers
|   |       `-- sd
|   |-- serio
|   |   |-- devices
|   |   `-- drivers
|   `-- usb
|       |-- devices
|       |   |-- 1-0:1.0 -> ../../../devices/platform/s3c2410-ohci/usb1/1-0:1.0
|       |   `-- usb1 -> ../../../devices/platform/s3c2410-ohci/usb1
|       `-- drivers
|          |-- hub
|          |   `-- 1-0:1.0 -> ../../../../devices/platform/s3c2410-ohci/usb1/1-0:1.0
|          |-- usb
|          |   `-- usb1 -> ../../../../devices/platform/s3c2410-ohci/usb1
|          `-- usb-storage
|-- class
|   |-- backlight
|   |   `-- s3c2410-bl
|   |-- graphics
|   |   `-- fb0
|   |-- i2c-adapter
|   |   `-- i2c-0
|   |       `-- device -> ../../../devices/platform/s3c2440-i2c/i2c-0
|   |-- i2c-dev
|   |   `-- i2c-0
|   |       `-- device -> ../../../devices/platform/s3c2440-i2c
|   |-- input
|   |   |-- event0
|   |   |-- event1
|   |   |-- mice
|   |   `-- mouse0
|   |-- lcd
|   |   `-- s3c2410-lcd
|   |-- mem
|   |   |-- full
|   |   |-- kmem
|   |   |-- kmsg
|   |   |-- mem
|   |   |-- null
|   |   |-- port
|   |   |-- random
|   |   |-- urandom
|   |   `-- zero
|   |-- misc
|   |   |-- apm_bios
|   |   |-- psaux
```

```
|   |   `-- rtc
|   |-- mmc_host
|   |   `-- mmc0
|   |       `-- device -> ../../../devices/platform/s3c2410-sdi
|   |-- mtd
|   |   |-- mtd0
|   |   `-- mtd0ro
|   |-- net
|   |   |-- eth0
|   |   |   `-- statistics
|   |   `-- lo
|   |       `-- statistics
|   |-- scsi_device
|   |-- scsi_host
|   |-- sound
|   |   |-- dsp
|   |   `-- mixer
|   |-- tty
|   |   |-- console
|   |   |-- ptmx
|   |   |-- s3c2410_serial0
|   |   |   `-- device -> ../../../devices/platform/s3c2440-uart.0
|   |   |-- s3c2410_serial1
|   |   |   `-- device -> ../../../devices/platform/s3c2440-uart.1
|   |   |-- s3c2410_serial2
|   |   |   `-- device -> ../../../devices/platform/s3c2440-uart.2
|   |   |-- tty
|   |   |-- tty0
|   |   |-- tty1
|   |   |-- tty10
|   |   |-- tty11
|   |   |-- tty12
|   |   |-- tty13
|   |   |-- tty14
|   |   |-- tty15
|   |   |-- tty16
|   |   |-- tty17
|   |   |-- tty18
|   |   |-- tty19
|   |   |-- tty2
|   |   |-- tty20
|   |   |-- tty21
|   |   |-- tty22
|   |   |-- tty23
|   |   |-- tty24
|   |   |-- tty25
|   |   |-- tty26
|   |   |-- tty27
|   |   |-- tty28
|   |   |-- tty29
|   |   |-- tty3
|   |   |-- tty30
|   |   |-- tty31
|   |   |-- tty32
|   |   |-- tty33
|   |   |-- tty34
|   |   |-- tty35
|   |   |-- tty36
|   |   |-- tty37
|   |   |-- tty38
|   |   |-- tty39
|   |   |-- tty4
|   |   |-- tty40
|   |   |-- tty41
|   |   |-- tty42
|   |   |-- tty43
|   |   |-- tty44
|   |   |-- tty45
|   |   |-- tty46
|   |   |-- tty47
```

```
|   |   |-- tty48
|   |   |-- tty49
|   |   |-- tty5
|   |   |-- tty50
|   |   |-- tty51
|   |   |-- tty52
|   |   |-- tty53
|   |   |-- tty54
|   |   |-- tty55
|   |   |-- tty56
|   |   |-- tty57
|   |   |-- tty58
|   |   |-- tty59
|   |   |-- tty6
|   |   |-- tty60
|   |   |-- tty61
|   |   |-- tty62
|   |   |-- tty63
|   |   |-- tty7
|   |   |-- tty8
|   |   `-- tty9
|   |-- usb
|   |-- usb_host
|   |   `-- usb1
|   |       `-- device -> ../../../devices/platform/s3c2410-ohci
|   `-- vc
|       |-- vcs
|       `-- vcsa
|-- devices
|   |-- platform
|   |   |-- power
|   |   |-- s3c2410-bl
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-bl
|   |   |   `-- power
|   |   |-- s3c2410-buttons
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-buttons
|   |   |   `-- power
|   |   |-- s3c2410-iis
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   `-- power
|   |   |-- s3c2410-lcd
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-lcd
|   |   |   `-- power
|   |   |-- s3c2410-ohci
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-ohci
|   |   |   |-- power
|   |   |   `-- usb1
|   |   |       |-- 1-0:1.0
|   |   |       |   |-- bus -> ../../../../../bus/usb
|   |   |       |   |-- driver -> ../../../../../bus/usb/drivers/hub
|   |   |       |   `-- power
|   |   |       |-- bus -> ../../../../bus/usb
|   |   |       |-- driver -> ../../../../bus/usb/drivers/usb
|   |   |       `-- power
|   |   |-- s3c2410-rtc
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-rtc
|   |   |   `-- power
|   |   |-- s3c2410-sdi
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-sdi
|   |   |   |-- mmc0:a95c
|   |   |   |   |-- block -> ../../../../block/mmcblk0
|   |   |   |   |-- bus -> ../../../../bus/mmc
|   |   |   |   |-- driver -> ../../../../bus/mmc/drivers/mmcblk
|   |   |   |   `-- power
```

```
|   |   |   `-- power
|   |   |-- s3c2410-ts
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-ts
|   |   |   `-- power
|   |   |-- s3c2410-usbgadget
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2410-usbgadget
|   |   |   `-- power
|   |   |-- s3c2410-wdt
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   `-- power
|   |   |-- s3c2440-i2c
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-i2c
|   |   |   |-- i2c-0
|   |   |   |   `-- power
|   |   |   `-- power
|   |   |-- s3c2440-nand
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-nand
|   |   |   `-- power
|   |   |-- s3c2440-sound
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-sound
|   |   |   `-- power
|   |   |-- s3c2440-uart.0
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-uart
|   |   |   `-- power
|   |   |-- s3c2440-uart.1
|   |   |   |-- bus -> ../../../bus/platform
|   |   |   |-- driver -> ../../../bus/platform/drivers/s3c2440-uart
|   |   |   `-- power
|   |   `-- s3c2440-uart.2
|   |       |-- bus -> ../../../bus/platform
|   |       |-- driver -> ../../../bus/platform/drivers/s3c2440-uart
|   |       `-- power
|   `-- system
|       |-- cpu
|       |   `-- cpu0
|       |-- s3c2440-core
|       |   `-- s3c2440-core0
|       |-- s3c24xx-dma
|       |   |-- s3c24xx-dma0
|       |   |-- s3c24xx-dma1
|       |   |-- s3c24xx-dma2
|       |   `-- s3c24xx-dma3
|       `-- timer
|           `-- timer0
|-- firmware
|-- kernel
|-- module
|   |-- lockd
|   |   `-- parameters
|   |-- mmc_block
|   |   `-- parameters
|   |-- nbd
|   |   `-- parameters
|   |-- scsi_mod
|   |   `-- parameters
|   |-- tcp_bic
|   |   `-- parameters
|   |-- usb_storage
|   |   `-- parameters
|   `-- usbcore
|       `-- parameters
`-- power
```

15.4. 카드제거에 따른 hotplug 동작

```
root@godori:~# ghc: interrupt occur **********************
ghc: interrupt occur **********************
ghc: interrupt occur **********************
ghc: interrupt occur **********************
MMC: sd_app_op_cond timed out. Probably no SD-Card here.
 5.MMC     : mmc_setup, mmc_discover_cards(SD) call
 5.MMC     : mmc_setup, mmc_discover_cards(MMC) call
 8.BUS     : bus_remove_device() bus: mmc, remove device: mmc0:a95c
10.GENHD   : block_hotplug: ktype = part , minor = 1
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: physdev->bus->name = mmc
10.GENHD   : block_hotplug: ktype = block, minor = 0
10.GENHD   : block_hotplug: physdev->bus->name pre
10.GENHD   : block_hotplug: physdev->bus->name = mmc
 1.CORE    : dev_hotplug: dev->bus_id: mmc0:a95c
 1.CORE    : dev_hotplug: hotplug --> call
 6.MMCSYSFS: mmc_bus_hotplug run
```

15.5. user level에서의 hotplug

15.5.1. SD card 삽입

```
// MMC hotplug발생( devices_subsystem )
default.hotplug: arguments (mmc) env (MMC_OEMID=5344 DEBUG=yes ACTION=add MMC_NAME=SD512
OLDPWD=/ HOME=/ SEQNUM=214 DEVPATH=/devices/platform/s3c2410-sdi/mmc0:a95c SUBSYSTEM=mmc
PATH=/bin:/sbin:/usr/sbin:/usr/bin MMC_MANFID=000003 PHYSDEVBUS=mmc MMC_CCC=101011111000
PWD=/etc/hotplug)

// block device driver hotplug발생, 2가지로 발생되는데 하나는 gendisk에 대한것이고, 나머지는 partition에
대한 것이다.
default.hotplug: arguments (block) env (DEBUG=yes ACTION=add OLDPWD=/ HOME=/ SEQNUM=215
MAJOR=254 DEVPATH=/block/mmcblk0 SUBSYSTEM=block PATH=/bin:/sbin:/usr/sbin:/usr/bin MINOR=0
PHYSDEVPATH=/devices/platform/s3c2410-sdi/mmc0:a95c PHYSDEVDRIVER=mmcblk PHYSDEVBUS=mmc
PWD=/etc/hotplug)

default.hotplug: arguments (block) env (DEBUG=yes ACTION=add OLDPWD=/ HOME=/ SEQNUM=216
MAJOR=254 DEVPATH=/block/mmcblk0/mmcblk0p1 SUBSYSTEM=block PATH=/bin:/sbin:/usr/sbin:/usr/bin
MINOR=1 PHYSDEVPATH=/devices/platform/s3c2410-sdi/mmc0:a95c PHYSDEVDRIVER=mmcblk
PHYSDEVBUS=mmc PWD=/etc/hotplug)
```

15.5.2. SD card 제거

```
// block device hotplug 처리
// partition에 대한 hotplug처리
default.hotplug[419]: arguments (block) env (DEBUG=yes ACTION=remove OLDPWD=/ HOME=/ SEQNUM=217
MAJOR=254 DEVPATH=/block/mmcblk0/mmcblk0p1 SUBSYSTEM=block PATH=/bin:/sbin:/usr/sbin:/usr/bin
MINOR=1 PHYSDEVPATH=/devices/platform/s3c2410-sdi/mmc0:a95c PHYSDEVDRIVER=mmcblk
PHYSDEVBUS=mmc PWD=/etc/hotplug)

// gendisk에 대한 hotplug처리
default.hotplug[420]: arguments (block) env (DEBUG=yes ACTION=remove OLDPWD=/ HOME=/ SEQNUM=218
MAJOR=254 DEVPATH=/block/mmcblk0 SUBSYSTEM=block PATH=/bin:/sbin:/usr/sbin:/usr/bin MINOR=0
PHYSDEVPATH=/devices/platform/s3c2410-sdi/mmc0:a95c PHYSDEVDRIVER=mmcblk PHYSDEVBUS=mmc
PWD=/etc/hotplug)

// mmc driver에 대한 hotplug처리
default.hotplug[433]: arguments (mmc) env (MMC_OEMID=5344 DEBUG=yes ACTION=remove
MMC_NAME=SD512 OLDPWD=/ HOME=/ SEQNUM=219 DEVPATH=/devices/platform/s3c2410-sdi/mmc0:a95c
SUBSYSTEM=mmc PATH=/bin:/sbin:/usr/sbin:/usr/bin MMC_MANFID=000003 PHYSDEVBUS=mmc
MMC_CCC=101011111000 PWD=/etc/hotplug)
```