==========

현재까지 발견된 이 책(**코드로 알아보는 ARM 리눅스 커널**) 1쇄본의 오탈자 정보와 오류 사항 그리고 보다 매끄러운 문장을 위해 수정한 내용들입니다. 불편을 끼쳐드려 죄송합니다.

혹시 이 외의 오탈자 정보를 발견하시는 분이 계시면 출판사 메일 jeipub골뱅이gmail.com이나 저자들 구글그룹 inux-kernel-arm@googlegroups.com으로 연락주시면 고맙겠습니다.

최종수정일자: 2012년 9월 17일
1쇄본 오탈자

아래의 오탈자 사항은 추후 재쇄 시에는 모두 반영하도록 하겠습니다. 집필과 편집 시에 미처 확인을 하지 못해 불편을 끼쳐드려 다시 한 번 죄송하다는 말씀을 드립니다.
==========

책 전체에서 아래 단어 변경
워크 큐 ➔ 워크큐

XXII페이지 4번째 줄
질핍후기 ➔ 집필 후기

30페이지 그림 3-5 아래에 아래의 출처 삽입
※ 참고문헌: 『ARM System-on-Chip Architecture』(Steve Furber 저, Addison-Wesley)

70페이지 그림 6-1(오른쪽 마지막 점선 박스에서)
WB ➔ Write Buffer

72페이지 첫 번째 문장에서
코드 섹션 ❶의 decompress_kernel 함수로 전달된 파라미터와 대응하는 레지스터의 값들은 코드 섹션 ❶의 decompress_kernel 함수로 전달된 파라미터와 대응하는 레지스터의 값들은 표 6-1과 같다.
➔
코드 섹션 ❶의 decompress_kernel 함수로 전달된 파라미터와 대응하는 레지스터의 값들은 표 6-1과 같다.

77페이지 오른쪽 맨 위
R2 : aTagPointer ➔ R2 : atags pointer

77페이지 가운데 열 위에서 세 번째 박스에서
proc.info.init 세션 ➔ proc.info.init 섹션

79페이지 두 번째 문단 첫 번째 줄
이 코드는 struct proct_info_list ➔ 이 코드는 struct proc_info_list

82페이지 아래에서 세 번째 문단 세 번째 줄
cpuval과 ➔ cpu_val과

86페이지 4번째 줄
레이블 3는 ➔ 레이블 3은

112페이지 그림 8-1에서
init 프로세스가 ➔ init 태스크가

120페이지 11번째 줄에서
init 프로세스가 ➔ init 태스크가

121페이지 첫 번째 줄에서
init task가 ➔ init 태스크가

121페이지 7번째 줄에서
init 프로세스가 ➔ init 태스크가

122페이지 6번째 줄에서
init task의 ➔ init 태스크의

128페이지 5번째 줄에서
지시자에 대해 집고 ➔ 지시자에 대해 짚고

136페이지 알아봅시다에서
Upate ➔ Update

154페이지 코드 12-2 8번째 줄(코드 빈 줄 포함)에서
reboot_se₩up("s"); ➔ reboot_setup("s");

166페이지 코드 12-7 아래로 세 번째 줄에서
(시스템에 존재하는 CPU 개수) ➔ (시스템이 지원할 수 있는 최대 CPU 개수)

168페이지 첫 번째 줄에서
0x0000FFFF ➔ 0xFFFF0000

200페이지 3번째 줄에서
cpu ID ➔ CPU ID

221페이지 그림 14-8의 첫 번째 줄에서 닫힘 괄호를 추가해야 함
virt_to_page(kaddr ➔ virt_to_page(kaddr)

228페이지 그림 15-3 우측 아래 7번에서
sruct page* (Page Array)
➔
struct page* (Page Array)

246페이지 15.5절 3번째 줄에서
가상주소 _stext와 end를 ➔ 가상주소 _stext와 _end를

264페이지 그림 16-1에서 mm_init_owner에 대한 박스 설명에서
mm_strucu가 ➔ mm_struct가

268p 2번째 줄
8MB ==> 8KB

269p 3번째 줄
8MB 영역은 ==> 8KB 영역은

281페이지 코드 16-9 제목을 한 줄로
코드 16-9 include/linux/preempt.h의 preempt_enable() 및 preempt_disable()

307페이지 그림 17-4 우측 하단 7번 설명에서
sruct page* ➔ struct page*

347페이지 코드 박스 제목 형식 수정
:: 코드 kernel/time/ntp.c
➔
:: kernel/time/ntp.c

358페이지 두 번째 줄에서
일이다 앞서 ➔ 일이다. 앞서

381페이지 3-4번째 줄에서
오더를 검사하지 그 외 페이지는
➔
오더를 검사하기 때문이다. 그 외 페이지는

386페이지 21.2절 첫 번째 줄에서

만약 CPU가 온라인에서 오프라인으로 변경될 때
➔
CPU가 온라인에서 오프라인으로 변경될 때

417페이지 코드 들여쓰기

```
for (i = 0; i < NUM_INIT_LISTS; i++) {
        kmem_list3_init(&initkmem_list3[i]);
        if (i < MAX_NUMNODES)
            cache_cache.nodelists[i] = NULL;
    }
```

➔

```
for (i = 0; i < NUM_INIT_LISTS; i++) {
    kmem_list3_init(&initkmem_list3[i]);
    if (i < MAX_NUMNODES)
        cache_cache.nodelists[i] = NULL;
}
```

422페이지 마지막 문단 첫 번째 줄에서
코드 섹션 ❶의 count 멤버는
➔
코드 섹션 ❶의 count 멤버변수는

422페이지 마지막 문단 두 번째 줄에서
코드 섹션 ❶의 high 멤버는
➔
코드 섹션 ❶의 high 멤버변수는

449페이지 코드 24-1 아래 문단 4번째 줄에서
살펴본 22장의 mem_init( ) ➔ 살펴본 20장의 mem_init( )

452페이지 첫 번째 줄에서
블록 디바이스드와 ➔ 블록 디바이스와

453페이지 그림 24-2 4번째 박스 내

```
static struct shrinker icache_shrinker = {
    .shrink = shrink_icache_memory,
    .seeks = DEFAULT_SEEKS,
};
```

➔

```
static struct shrinker dcache_shrinker = {
    .shrink = shrink_dcache_memory,
```

```
    .seeks = DEFAULT_SEEKS,
};
```

458페이지 밑에서 네 번째 문단 3번째 줄에서
여러 “제한” 사항을 관리한다. ➔ 여러 제한사항을 관리한다.

459페이지 6번째 줄에서
워크 큐(work queue)에 ➔ 워크큐(workqueue)에

464-465페이지 코드 24-6 (코드 섹션 번호 위치 오류)
〈기존〉

```
int _ _init sysfs_init(void)
{
    ...
    sysfs_dir_cachep = kmem_cache_create(“sysfs_dir_cache”,
                                         sizeof(struct sysfs_dirent),
                                         0, 0, NULL);
    ...

    err = sysfs_inode_init();                                    ❶
    ...
    err = register_filesystem(&sysfs_fs_type);                   ❷
    if (!err) {
        sysfs_mount = kern_mount(&sysfs_fs_type);                ❸
        ...
    } else
        goto out_err;

    ...
}
```

〈수정〉

```
int _ _init sysfs_init(void)
{
    ...
    sysfs_dir_cachep = kmem_cache_create(“sysfs_dir_cache”,      ❶
                                         sizeof(struct sysfs_dirent),
                                         0, 0, NULL);
    ...

    err = sysfs_inode_init();                                    ❷
```

```
    ...

    err = register_filesystem(&sysfs_fs_type);                    ❸
    if (!err) {
        sysfs_mount = kern_mount(&sysfs_fs_type);                 ❹
        ...
    } else
        goto out_err;

    ...
}
```

466페이지 코드 24-7 (코드 섹션 번호 위치 및 들여쓰기 간격 조정)

```
void _ _init bdev_cache_init(void)
{
    int err;
    struct vfsmount *bd_mnt;
    bdev_cachep = kmem_cache_create("bdev_cache", sizeof(struct bdev_inode),
            0, (SLAB_HWCACHE_ALIGN|SLAB_RECLAIM_ACCOUNT|
            SLAB_MEM_SPREAD|SLAB_PANIC),
            init_once);                                           ❶
```

<수정>

```
void __init bdev_cache_init(void)
{
    int err;
    struct vfsmount *bd_mnt;

    bdev_cachep = kmem_cache_create("bdev_cache", sizeof(struct bdev_inode),  ❶
                    0, (SLAB_HWCACHE_ALIGN|SLAB_RECLAIM_ACCOUNT|
                            SLAB_MEM_SPREAD|SLAB_PANIC),
                    init_once);
```

470페이지 첫 번째 코드 박스 제목 수정

kernel/signal.c의 signals_init()

➔

코드 25-2 kernel/signal.c의 signals_init()

476페이지 그림 25-2에서 (콜론 사이 공백 필요)

nlmsghdr: netlink

➔

nlmsghdr : netlink

477페이지 25.7.2절 제목 수정
25.7.2 dealyacct_init() ➔ 25.7.2 delayacct_init()

477페이지 4번째 문단 첫 번째 줄에서
예를 들면, 태스크가 수행하기 위해
➔
예를 들면, 태스크를 수행하기 위해

478페이지 마지막 문단 첫 번째 줄에서
dealyacct_tsk_init( ) 함수에
➔
delayacct_tsk_init( ) 함수에

480페이지 알아봅시다 마지막 문장
pid = 1에 대한 ➔ PID = 1에 대한

484페이지 밑에서 4번째 줄에서
start_one_pdflush_thread 함수는
➔
start_one_pdflush_thread() 함수는

488페이지 코드 박스 아래로 2번째 줄
my_work를 pdflush_work 구조체를
➔
my_work를 pdflush_work 구조체의

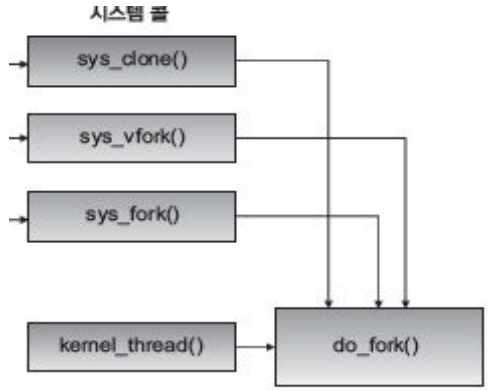
490페이지 26.5.1절 2번째 줄
wb_timer_fin( ) 함수는
➔
wb_timer_fn( ) 함수는

491페이지 2번째 줄
어떻게 백킹 스토어인 블록 디바이스에 써야
➔
백킹 스토어인 블록 디바이스에 어떻게 쓰여져야

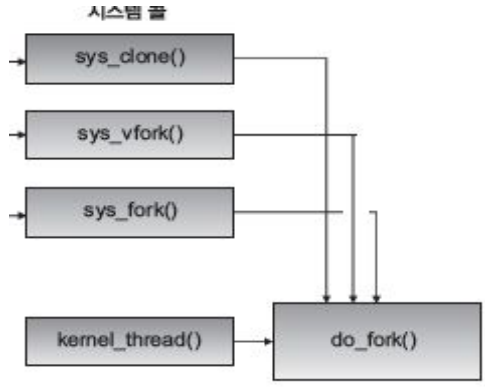
492페이지 알아봅시다 첫 번째 줄
vm_stat는 ➔ vm_stat은

497페이지 마지막 줄
코드 섹션 (2)는 현제 ➔ 코드 섹션 (2)는 현재

500페이지 그림 28-1 화살표 나란하게 정리
〈기존〉



〈수정〉



500페이지 아래에서 두 번째 문단 첫 번째 줄에서
유저 레벨에서 fork, vfork, clone, pthread_create 등의
➔
유저 레벨에서 fork( ), vfork( ), clone( ), pthread_create( ) 등의

506페이지 각주 89번 세 번째 줄에서
ntp daemon의 경우 네트워크와 관련되지 않은 권한은 제한하여 해킹이 되도
➔ ntp 데몬의 경우 네트워크와 관련되지 않은 권한은 제한하여 해킹이 되어도

522페이지 첫 번째 줄에서
kernel_init다. ➔ kernel_init()이다.

522페이지 코드 30-1의 코드 사이에 줄 간격 수정
〈기존〉

```
static int _ _init kernel_init(void * unused)
{
    lock_kernel();

    set_cpus_allowed_ptr(current, cpu_all_mask);                    (1)
    init_pid_ns.child_reaper = current;                             (2)
    cad_pid = task_pid(current);                                    (3)
    smp_prepare_cpus(setup_max_cpus);                               (4)
    do_pre_smp_initcalls();                                         (5)
    start_boot_trace();
    smp_init();                                                     (6)
    sched_init_smp();                                               (7)
    do_basic_setup();                                               (8)
    if (!ramdisk_execute_command)                                   (9)
        ramdisk_execute_command = "/init";
    if (sys_access((const char _ _user *) ramdisk_execute_command, 0) != 0) {
        ramdisk_execute_command = NULL;
        prepare_namespace();
    }
    init_post();                                                    (10)
    return 0;
}
```

<수정>

```
static int __init kernel_init(void * unused)
{
    lock_kernel();

    set_cpus_allowed_ptr(current, cpu_all_mask);                    (1)

    init_pid_ns.child_reaper = current;                             (2)

    cad_pid = task_pid(current);                                    (3)

    smp_prepare_cpus(setup_max_cpus);                               (4)

    do_pre_smp_initcalls();                                         (5)
    start_boot_trace();

    smp_init();                                                     (6)
    sched_init_smp();                                               (7)
```

```
    do_basic_setup();                                                    (8)

    if (!ramdisk_execute_command)                                        (9)
        ramdisk_execute_command = "/init";

    if (sys_access((const char __user *) ramdisk_execute_command, 0) != 0) {
        ramdisk_execute_command = NULL;
        prepare_namespace();
    }

    init_post();                                                         (10)
    return 0;
}
```

522페이지 코드 30-1 아래 첫 번째 문단 세 번째 줄(2쇄에서는 524페이지 세 번째 줄로 이동)
32.3절을 참조하자. ➔ 31.4절을 참조하자.

523페이지 두 번째 줄에서(2쇄는 세 번째 문단 첫 번째 줄에서 반영되었음)
유저가 ctrl-alt-del을 동시에 누를 경우 ctrl_alt_del() 함수가 호출이 된다.
➔
유저가 Ctrl+Alt+Del 키를 동시에 누를 경우 ctrl_alt_del() 함수가 호출된다.

523페이지 5번째 문단 3번째 줄에서
isolcpu(isolated cpu)로 지정되지 않은 cpu 중
➔
isolcpu(isolated CPU)로 지정되지 않은 CPU 중

523페이지 7번째 문단 첫 번째 줄에서
/init로 설정한다. ➔ "/init"으로 설정한다.

523페이지 8번째 문단 두 번째 줄에서
기본 파일 디스크립터인 0, 1, 2를 만들고
➔
표준 입력, 표준 출력, 표준 에러에 해당하는 파일 디스크립터를 만들고

525페이지 각주 95번에서
sched_init()에서 아래와 같이 SCHED_SOFTIRQ가 발생할 때 호출할 함수로
➔
sched_init()에서 SCHED_SOFTIRQ가 발생할 때 호출될 함수로

530페이지 3-6번째 줄에서
init_cpu_workqueue, create_workqueue_thread, start_workqueue_thread를 차례로 호출한다.
➔
init_cpu_workqueue( ), create_workqueue_thread( ), start_workqueue_thread( )를 차례로 호출한다.

순회하면서 init_cpu_workqueue, create_workqueue_thread, start_workqueue_thread를 호출한다.
➔
순회하면서 init_cpu_workqueue( ), create_workqueue_thread( ), start_workqueue_thread( )를 호출한다.

532페이지 첫 번째 줄에서
create_workqueue_thread를 호출하는데, ➔ create_workqueue_thread( )를 호출하는데,

532페이지 5번째 줄에서
이렇게 해서 생성한 워크 스레드는 ➔ 이렇게 해서 생성한 워커 스레드는

533페이지 2, 4, 5행에서
kthread_bind 함수를 ➔ kthread_bind( ) 함수를
전반적으로 workqueue를 ➔ 전반적으로 워크큐를
workqueue의 실질적인 수행을 담당하는 함수는 worker_thread다.
➔
워크큐의 실질적인 수행을 담당하는 함수는 worker_thread( )다.

533페이지 마지막 줄에서
워크 스레드가 깨어나게 된다. ➔ 워커 스레드가 깨어나게 된다.

534페이지 첫 번째 줄에서
워크 스레드가 깨어나면 ➔ 워커 스레드가 깨어나면

535페이지 두 번째 줄에서
워크 스레드에 등록함으로써 ➔ 워커 스레드에 등록함으로써

536페이지 마지막 문단 마지막 줄
워크 스레드에 의해 ➔ 워커 스레드에 의해

539페이지 코드 박스 아래로 6번째 줄에서
공통적인 속성들로 구성시킨다. ➔ 공통적인 속성들로 구성된다.

539페이지 마지막 문단 첫 번째 줄에서
수많은 kboject 중에서 서로 연관된 kboject를
➔
수많은 kobject 중에서 서로 연관된 kobject를

540페이지 두 번째, 세 번째 문단에서
release는 kboject의 ➔ release는 kobject의
sysfs 상에 kboject가 ➔ sysfs 상에 kobject가
이때 kboject의 dentry 포인터가 ➔ 이때 kobject의 dentry 포인터가

541페이지 그림 30-7(가운데 사각형 내)
.defautl_attrs[] ➔ .default_attrs[]

543페이지 세 번째 줄
kset_create_and_add는 아래와 ➔ kset_create_and_add( )는 아래와

548페이지 마지막 문단에서
init_irq_proc에서는 ➔ init_irq_proc( )에서는
init_irq_proc 함수 흐름도는 ➔ init_irq_proc( ) 함수 흐름도는

548페이지 그림 30-9 제목에서
그림 30-9 init_irq_proc 함수 흐름도 ➔ 그림 30-9 init_irq_proc( ) 함수 흐름도

549페이지 두 번째, 세 번째 문단에서
proc_mkdir을 호출하여 ➔ proc_mkdir( )을 호출하여
init_irq_proc 함수의 핵심은 _ _proc_create와 proc_register를 호출하여
➔
init_irq_proc( ) 함수의 핵심은 _ _proc_create( )와 proc_register( )를 호출하여
_ _proc_create 함수의 기능은 ➔ _ _proc_create( ) 함수의 기능은
_ _proc_create를 통해서 ➔ _ _proc_create( )를 통해서

549페이지 마지막 문단 첫 번째 줄에서
do_initcalls은 아래에서 ➔ do_initcalls( )는 아래에서

550페이지 3번째 줄에서
main 함수를 ➔ 메인 함수를

550페이지 5번째 줄에서
do_initcalls에 의해서 ➔ do_initcalls( )에 의해서

551페이지 7번째 줄에서

또한 *initcall 계열의 ➔ 또한 initcall 계열의

552페이지 5-6번째 줄에서
do_one_initcall을 호출한다. do_one_initcall 함수는 아래에 나타내었다.
➔
do_one_initcall( )을 호출한다. do_one_initcall( ) 함수는 아래에 나타내었다.

554페이지 첫 번째 문단
do_one_initcall의 수행 과정을 ➔ do_one_initcall( )의 수행 과정을
아직 do_initcalls 함수가 ➔ 아직 do_initcalls( ) 함수가
flush_scheduled_work 함수를 ➔ flush_scheduled_work( ) 함수를
이로써 do_initcalls 함수가 ➔ 이로써 do_initcalls( ) 함수가

554페이지 30.3절 두 번째 줄에서
kernel_init을 위한 커널 스레드를 생성하였다. 여기 init_post 함수에서는
➔
kernel_init( )을 위한 커널 스레드를 생성하였다. 여기 init_post( ) 함수에서는

559페이지 4번째 줄에서
kthread_create_list에서 생성할 태스크에 대한 멤버에 대한 정보를
➔
kthread_create_list에서 생성할 태스크 멤버에 대한 정보를

561페이지 그림 31-1에서
좌측 하단 struct k_sigaction 부분 글자 'o' 위에 겹쳐진 사각형 제거

563페이지 4번째 문단 세 번째 줄에서
우선순위(priority) 값으로 변환한다. 태스크의 static_prio 값을 채우고 종료한다.
➔
우선순위(priority) 값으로 변환한 후 태스크의 static_prio 변수에 값을 채우고 종료한다.

576페이지 6번째 줄에서
hlist head (pid_hash) 구조체 ➔ hlist_head (pid_hash) 구조체

576페이지 7번째 줄에서
pid를 가지고 어떻게 ➔ PID를 가지고 어떻게

646페이지
워크 스레드 ➔ 워커 스레드

651페이지

cgroup_init_early() 118-120

➔

cgroup_init_early() 118-123

660페이지

mem_init( ) 364-366 ➔ mem_init( ) 363-366