Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

#41

강사 : Innova Lee(이 상훈)

학생 : 김 시윤

1.배운내용 복습.

커널 드라이빙

선생님께서 손수 제작하신 monitor_hack.c
make 하면 여러 디바이스 파일들이 생김

디바이스 적용
sudo insmod monitor_hack.ko
->이때 동작하는 함수 module_init(syscall_hooking_init);

오늘은 insmod 했을 때 동작하는 이닛 함수를 살펴보았다.

따라서 우리는 syscall_hooking_init 함수를 살펴본다.

syscall_hooking_init 함수

```
int syscall_hooking_init(void)
{
    unsigned long cr0;

    if((sys_call_table = locate_sys_call_table()) == NULL)
    {
        printk("<0>Can't find sys_call_table\n");
        return -1;
    }

    printk("<0>sys_call_table is at[%p]\n", sys_call_table);

    // CR0 레지스터를 읽어옴
    cr0 = read_cr0();
    // Page 쓰기를 허용함
    write_cr0(cr0 & ~0x00010000);

    /* set_memory_rw 라는 심볼을 찾아와서 fixed_set_memory_rw 에 설정함 */
    fixed_set_memory_rw = (void *)kallsyms_lookup_name("set_memory_rw");
    if(!fixed_set_memory_rw)
    {
        printk("<0>Unable to find set_memory_rw symbol\n");
        return 0;
    }

    /* 시스템 콜 테이블이 위치한 물리 메모리에 읽고 쓰기 권한 주기 */
    fixed_set_memory_rw(PAGE_ALIGN((unsigned long)sys_call_table) - PAGE_SIZE, 3);

    orig_call = (void *)sys_call_table[__NR_open];
    sys_call_table[__NR_open] = (void *)sys_our_open;
    write_cr0(cr0);
    printk("<0>Hooking Success!\n");
    return 0;
}
```

처음 보이는 cr0 레지스터, 인텔 CPU에 있는 레지스터로 데이터시트를 확인한다.

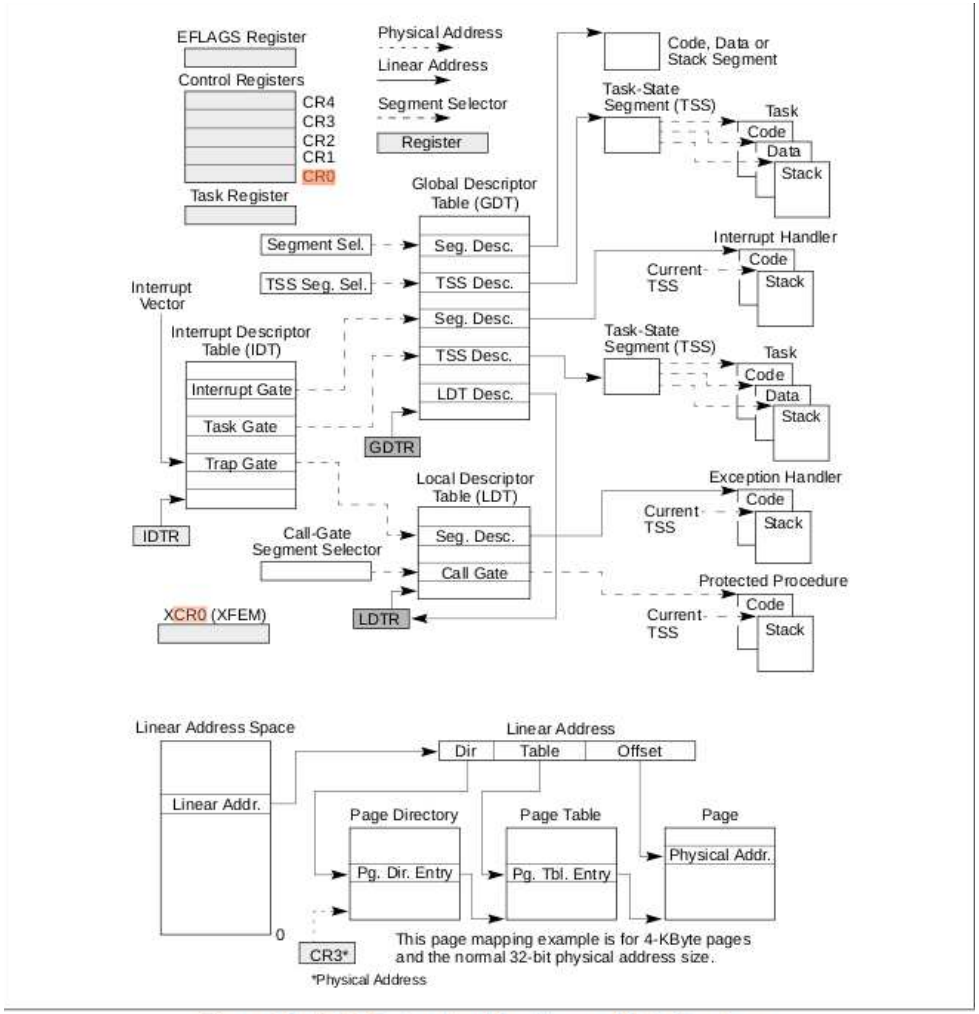
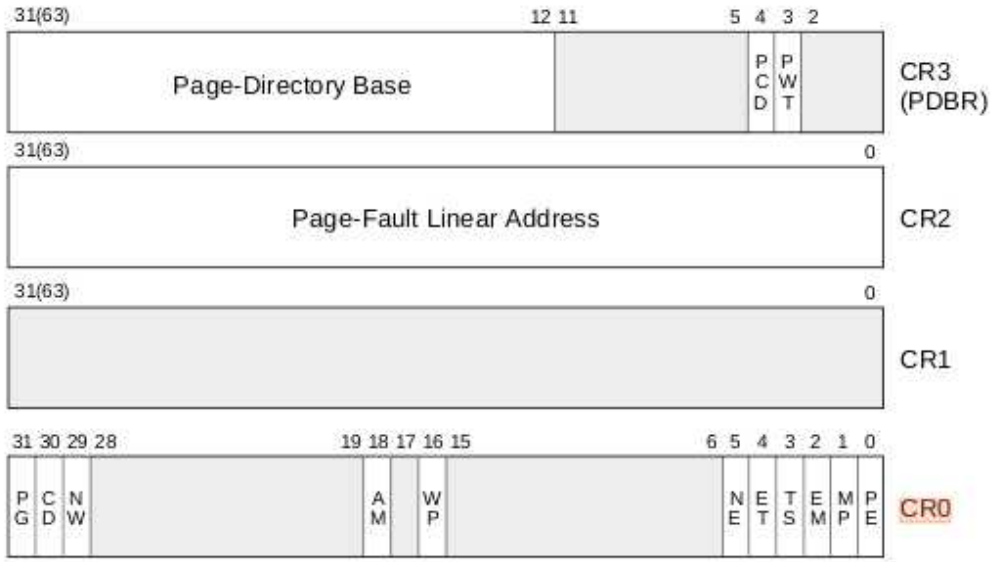


Figure 2-1. IA-32 System-Level Registers and Data Structures

시스템레벨에서 레지스터의 흐름도 여기서 cr0 는 여러개 컨트롤 레지스터들 중 0번째를 의미한다.

즉 시피유를 통제하는 역할



컨트롤 레지스터들은 각각 통제할수 있는 영역이 다르다.

우리는 그중 cr0의 페이지 읽고 쓰기를 볼 것이다.

PG **Paging (bit 31 of CR0)** — Enables paging when set; disables paging when clear. When paging is disabled, all linear addresses are treated as physical addresses. The PG flag has no effect if the PE flag (bit 0 of register CR0) is not also set; setting the PG flag when the PE flag is clear causes a general-protection exception (#GP). See also: Chapter 4, "Paging."

On Intel 64 processors, enabling and disabling IA-32e mode operation also requires modifying CR0.PG.

CD **Cache Disable (bit 30 of CR0)** — When the CD and NW flags are clear, caching of memory locations for the whole of physical memory in the processor's internal (and external) caches is enabled. When the CD flag is set, caching is restricted as described in Table 11-5. To prevent the processor from accessing and updating its caches, the CD flag must be set and the caches must be invalidated so that no cache hits can occur.

See also: Section 11.5.3, "Preventing Caching," and Section 11.5, "Cache Control."

NW **Not Write-through (bit 29 of CR0)** — When the NW and CD flags are clear, write-back (for Pentium 4, Intel Xeon, P6 family, and Pentium processors) or write-through (for Intel486 processors) is enabled for writes that hit the cache and invalidation cycles are enabled. See Table 11-5 for detailed information about the effect of the NW flag on caching for other settings of the CD and NW flags.

AM **Alignment Mask (bit 18 of CR0)** — Enables automatic alignment checking when set; disables alignment checking when clear. Alignment checking is performed only when the AM flag is set, the AC flag in the EFLAGS register is set, CPL is 3, and the processor is operating in either protected or virtual-8086 mode.

WP **Write Protect (bit 16 of CR0)** — When set, inhibits supervisor-level procedures from writing into read-only pages; when clear, allows supervisor-level procedures to write into read-only pages (regardless of the U/S bit setting; see Section 4.1.3 and Section 4.6). This flag facilitates implementation of the copy-on-write method of creating a new process (forking) used by operating systems such as UNIX.

NE **Numeric Error (bit 5 of CR0)** — Enables the native (internal) mechanism for reporting x87 FPU errors when set; enables the PC-style x87 FPU error reporting mechanism when clear. When the NE flag is clear and the IGNNE# input is asserted, x87 FPU errors are ignored. When the NE flag is clear and the IGNNE# input is deasserted, an unmasked x87 FPU error causes the processor to assert the FERR# pin to generate an external interrupt and to stop instruction execution immediately before executing the next waiting floating-point instruction or WAIT/FWAIT instruction.

The FERR# pin is intended to drive an input to an external interrupt controller (the FERR# pin emulates the ERROR# pin of the Intel 287 and Intel 387 DX math coprocessors). The NE flag, IGNNE# pin, and FERR# pin are used with external logic to implement PC-style error reporting. Using FERR# and IGNNE# to handle floating-point exceptions is deprecated by modern operating systems; this non-native approach also limits newer processors to operate with one logical processor active.

See also: Section 8.7, "Handling x87 FPU Exceptions in Software" in Chapter 8, "Programming with the x87 FPU," and Appendix A, "EFLAGS Cross-Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

ET **Extension Type (bit 4 of CR0)** — Reserved in the Pentium 4, Intel Xeon, P6 family, and Pentium processors. In the Pentium 4, Intel Xeon, and P6 family processors, this flag is hardcoded to 1. In the Intel386 and Intel486 processors, this flag indicates support of Intel 387 DX math coprocessor instructions when set.

TS **Task Switched (bit 3 of CR0)** — Allows the saving of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4

이 여러개 옵션중 16번 Write Protect (레지스터 16번째비트) 옵션을 본다.
16번째 비트가 1일 때 쓰기 가 억제되고 0일 때 쓰기가 가능하다.

함수에서는 변수로서 cr0를 선언했기 때문에 cr0변수에 레지스터의 값을 넣어줘야 하기 때문에 cr0를 읽어주는 동작을 실행시킨다.

cr0 = read_cr0();

```
static inline unsigned long read_cr0(void)
{
    return native_read_cr0();
}
```

native_read_cr0를 반환한다. native_read_cr0를 확인해본다.

```
static inline unsigned long native_read_cr0(void)
{
    unsigned long val;
    // asm volatile(구동시킬 명령어 : 출력 : 입력 : 어셈블러 지시어)
    asm volatile("mov %%cr0,%0\n\t" : "=r" (val), "=m" (__force_order));
    return val;
}
```

확인해보니 어셈블리어(기계어) 코드가 존재한다.
mov %%cr0,%0
여기서 %0은 처음 선언된 변수를 가리킨다. 즉 %0 = val이라는 소리다.
다시 써보면
asm volatile("mov %%cr0,val\n\t" : "=r" (val), "=m" (__forve_order));
여기서는 ':' 가 1개밖에 없으므로 구동시킬 명령어를 읽어온다.

r = 레지스터 , m = 메모리

write_cr0(cr0 & ~0x00010000);
읽어온 cr0 의 16번째 비트를 0으로 만들어주어 페이지 쓰기를 허용시켰다.

/* set_memory_rw 라는 심볼을 찾아와서 fixed_set_memory_rw 에 설정함 */
fixed_set_memory_rw = (void *)kallsyms_lookup_name("set_memory_rw");
-> 이제부터 여기서 set_memory_rw를 찾아야 한다. kallsyms_lookup_name 은 심볼을 활성화시키는 역할을 하는녀석이다 보안차원에서 심볼은 kallsyms를 사용하지 않

을 경우 활성화 되지 않는다. 즉 권한을 얻을 수 없다.

```
siyun@siyun-CR62-6M: ~/kernel/linux-4.4
// change_page_attr_set_clr(addr, 3, 2, 0, 0, 0, NULL)
static int change_page_attr_set_clr(unsigned long *addr, int numpages,
                    pgprot_t mask_set, pgprot_t mask_clr,
                    int force_split, int in_flag,
                    struct page **pages)
{
    /*
    struct cpa_data {
        unsigned long   *vaddr;
        pgd_t       *pgd;
        pgprot_t    mask_set;
        pgprot_t    mask_clr;
        int     numpages;
        int     flags;
        unsigned long   pfn;
        unsigned    force_split : 1;
        int     curpage;
        struct page **pages;
    };
    */
    struct cpa_data cpa;
    int ret, cache, checkalias;
    unsigned long baddr = 0;

    /* 위 구조체 처럼 생긴 녀석을 싹 0 으로 초기화 시킴 */
    memset(&cpa, 0, sizeof(cpa));

    /*
     * Check, if we are requested to change a not supported
     * feature:
     */
    mask_set = canon_pgprot(mask_set);  // mask_set == 2
    mask_clr = canon_pgprot(mask_clr);  // mask_clr == 0
    if (!pgprot_val(mask_set) && !pgprot_val(mask_clr) && !force_split)
        return 0;

    /* Ensure we are PAGE_SIZE aligned */
    // in_flag == 0
    if (in_flag & CPA_ARRAY) {
        int i;
        for (i = 0; i < numpages; i++) {
            if (addr[i] & ~PAGE_MASK) {
                addr[i] &= PAGE_MASK;
                WARN_ON_ONCE(1);
            }
        }
    } else if (!(in_flag & CPA_PAGES_ARRAY)) {
        /*
         * in_flag of CPA_PAGES_ARRAY implies it is aligned.
         * No need to cehck in that case
         */

        /* 4096 단위로 정렬, 즉 페이지 단위로 정렬됨을 보장해줌 */
        if (*addr & ~PAGE_MASK) {
            *addr &= PAGE_MASK; // *addr = *addr & PAGE_MASK
            /*
             * People should not be passing in unaligned addresses:
             */
            WARN_ON_ONCE(1);
```

일단 리턴으로 chage_page_attr_clear을 해주고

```
siyun@siyun-CR62-6M: ~/kernel/linux-4.4
     */
    if (!(cpa.flags & CPA_FLUSHTLB))
        goto out;

    /*
     * No need to flush, when we did not set any of the caching
     * attributes:
     */
    cache = !!pgprot2cachemode(mask_set);

    /*
     * On success we use CLFLUSH, when the CPU supports it to
     * avoid the WBINVD. If the CPU does not support it and in the
     * error case we fall back to cpa_flush_all (which uses
     * WBINVD):
     */
    if (!ret && cpu_has_clflush) {
        if (cpa.flags & (CPA_PAGES_ARRAY | CPA_ARRAY)) {
            cpa_flush_array(addr, numpages, cache,
                    cpa.flags, pages);
        } else
            cpa_flush_range(baddr, numpages, cache);
    } else
        cpa_flush_all(cache);

out:
    return ret;
}

static inline int change_page_attr_set(unsigned long *addr, int numpages,
                       pgprot_t mask, int array)
{
    // change_page_attr_set_clr(addr, 3, 2, 0, 0, (array ? CPA_ARRAY : 0), NULL)
    // CPA_ARRAY == 2
    // change_page_attr_set_clr(addr, 3, 2, 0, 0, (array ? 2 : 0), NULL)
    // (∵ array == 0)
    // change_page_attr_set_clr(addr, 3, 2, 0, 0, 0, NULL)

    return change_page_attr_set_clr(addr, numpages, mask, __pgprot(0), 0,
        (array ? CPA_ARRAY : 0), NULL);
}

static inline int change_page_attr_clear(unsigned long *addr, int numpages,
                         pgprot_t mask, int array)
{
    return change_page_attr_set_clr(addr, numpages, __pgprot(0), mask, 0,
        (array ? CPA_ARRAY : 0), NULL);
}

static inline int cpa_set_pages_array(struct page **pages, int numpages,
                       pgprot_t mask)
{
    return change_page_attr_set_clr(NULL, numpages, mask, __pgprot(0), 0,
        CPA_PAGES_ARRAY, pages);
}

static inline int cpa_clear_pages_array(struct page **pages, int numpages,
                         pgprot_t mask)
{
                                                              1539,9-
```

그걸 다시 셋해주는데 4단계 페이징을 통해 물리메모리 주소를 획득 후 읽기 쓰기 권한을 셋해준다.

```
-CR62-6M: ~/kernel/linux-4.4
static int __change_page_attr(struct cpa_data *cpa, int primary)
{
    unsigned long address;
    int do_split, err;
    unsigned int level;
    pte_t *kpte, old_pte;

    /*
    vaddr = 시스템 콜 테이블 주소
    pages = NULL
    numpages = 3
    mask_set = 2
    mask_clr = 0
    flags = 0
    curpage = 0
    force_split = 0 */

    if (cpa->flags & CPA_PAGES_ARRAY) {
        struct page *page = cpa->pages[cpa->curpage];
        if (unlikely(PageHighMem(page)))
            return 0;
        address = (unsigned long)page_address(page);
    } else if (cpa->flags & CPA_ARRAY)
        address = cpa->vaddr[cpa->curpage];
    else
        address = *cpa->vaddr;
repeat:
    /* 4 단계 페이징을 거쳐 실제 물리 메모리 획득
       주의할 것은 이미 시스템 콜 테이블은 부팅시 초기화 되므로
       새로운 페이지를 할당한 것이 아니라
       이미 존재하는 페이지를 통해 물리 주소를 획득한 것임
       그러므로 kpte 는 명확한 값을 가지고 있음 */
    kpte = _lookup_address_cpa(cpa, address, &level);
    if (!kpte)
        return __cpa_process_fault(cpa, address, primary);

    old_pte = *kpte;
    if (!pte_val(old_pte))
        return __cpa_process_fault(cpa, address, primary);

    if (level == PG_LEVEL_4K) {
        pte_t new_pte;
        pgprot_t new_prot = pte_pgprot(old_pte);
        unsigned long pfn = pte_pfn(old_pte);

        pgprot_val(new_prot) &= ~pgprot_val(cpa->mask_clr);
        pgprot_val(new_prot) |= pgprot_val(cpa->mask_set);

        new_prot = static_protections(new_prot, address, pfn);

        /*
         * Set the GLOBAL flags only if the PRESENT flag is
         * set otherwise pte_present will return true even on
         * a non present pte. The canon_pgprot will clear
         * _PAGE_GLOBAL for the ancient hardware that doesn't
         * support it.
         */
        if (pgprot_val(new_prot) & _PAGE_PRESENT)
            pgprot_val(new_prot) |= _PAGE_GLOBAL;
        else
            pgprot_val(new_prot) &= ~_PAGE_GLOBAL;
```

```
R62-6M: ~/kernel/linux-4.4
        /*
         * Set the GLOBAL flags only if the PRESENT flag is
         * set otherwise pte_present will return true even on
         * a non present pte. The canon_pgprot will clear
         * _PAGE_GLOBAL for the ancient hardware that doesn't
         * support it.
         */
        if (pgprot_val(new_prot) & _PAGE_PRESENT)
            pgprot_val(new_prot) |= _PAGE_GLOBAL;
        else
            pgprot_val(new_prot) &= ~_PAGE_GLOBAL;

        /*
         * We need to keep the pfn from the existing PTE,
         * after all we're only going to change it's attributes
         * not the memory it points to
         */
        new_pte = pfn_pte(pfn, canon_pgprot(new_prot));
        cpa->pfn = pfn;
        /*
         * Do we really change anything ?
         */
        if (pte_val(old_pte) != pte_val(new_pte)) {
            set_pte_atomic(kpte, new_pte);
            cpa->flags |= CPA_FLUSHTLB;
        }
        cpa->numpages = 1;
        return 0;
    }

    /*
     * Check, whether we can keep the large page intact
     * and just change the pte:
     */
    do_split = try_preserve_large_page(kpte, address, cpa);
    /*
     * When the range fits into the existing large page,
     * return. cp->numpages and cpa->tlbflush have been updated in
     * try_large_page:
     */
    if (do_split <= 0)
        return do_split;

    /*
     * We have to split the large page:
     */
    err = split_large_page(cpa, kpte, address);
    if (!err) {
        /*
         * Do a global flush tlb after splitting the large page
         * and before we do the actual change page attribute in the PTE.
         *
         * With out this, we violate the TLB application note, that says
         * "The TLBs may contain both ordinary and large-page
         *  translations for a 4-KByte range of linear addresses. This
         *  may occur if software modifies the paging structures so that
         *  the page size used for the address range changes. If the two
         *  translations differ with respect to page frame or attributes
         *  (e.g., permissions), processor behavior is undefined and may
         *  be implementation-specific."
```

```
CR62-6M: ~/kernel/linux-4.4
                cpa->flags |= CPA_FLUSHTLB;
            }
            cpa->numpages = 1;
            return 0;
        }

        /*
         * Check, whether we can keep the large page intact
         * and just change the pte:
         */
        do_split = try_preserve_large_page(kpte, address, cpa);
        /*
         * When the range fits into the existing large page,
         * return. cp->numpages and cpa->tlbflush have been updated in
         * try_large_page:
         */
        if (do_split <= 0)
            return do_split;

        /*
         * We have to split the large page:
         */
        err = split_large_page(cpa, kpte, address);
        if (!err) {
            /*
             * Do a global flush tlb after splitting the large page
             * and before we do the actual change page attribute in the PTE.
             *
             * With out this, we violate the TLB application note, that says
             * "The TLBs may contain both ordinary and large-page
             *  translations for a 4-KByte range of linear addresses. This
             *  may occur if software modifies the paging structures so that
             *  the page size used for the address range changes. If the two
             *  translations differ with respect to page frame or attributes
             *  (e.g., permissions), processor behavior is undefined and may
             *  be implementation-specific."
             *
             * We do this global tlb flush inside the cpa_lock, so that we
             * don't allow any other cpu, with stale tlb entries change the
             * page attribute in parallel, that also falls into the
             * just split large page entry.
             */
            flush_tlb_all();
            goto repeat;
        }

        return err;
    }

    static int __change_page_attr_set_clr(struct cpa_data *cpa, int checkalias);

    static int cpa_process_alias(struct cpa_data *cpa)
    {
        struct cpa_data alias_cpa;
        unsigned long laddr = (unsigned long)__va(cpa->pfn << PAGE_SHIFT);
        unsigned long vaddr;
        int ret;

        if (!pfn_range_is_mapped(cpa->pfn, cpa->pfn + 1))
            return 0;
```

```
    orig_call = (void *)sys_call_table[__NR_open];
    sys_call_table[__NR_open] = (void *)sys_our_open;
    write_cr0(cr0);
```

그다음에 시스템 콜 테이블에 있는 원래의 NR_open을 갖고와 선생님께서 선언한 open변수에 저장하고 cr0권한을 실행시켜 우리가 마음데로 갖고 놀 수 있게 해준다.

아직 전체적인 큰 흐름 정도밖에 못하겠습니다.
지금은 드라이빙을 이곳 저곳 계속 들어가야 할 것 같습니다.
이곳 저곳 많이 돌아다니면서 스샷을 많이 찍고다니긴 했는데
설명이 가능한 것들만 첨부해서 올렸습니다..