R E P O R T

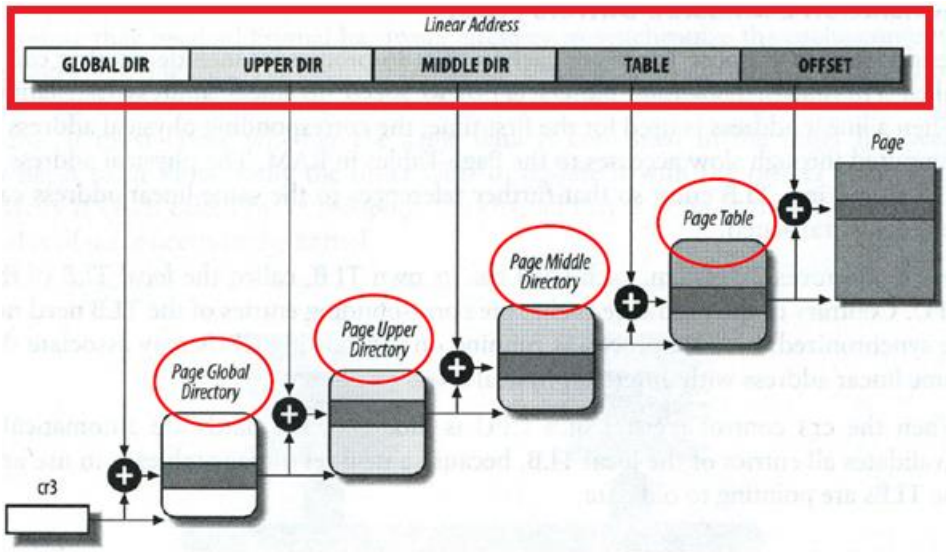
시스템 프로그래밍 과제 #2

학 과	**컴퓨터 과학**
교수님	**차호정**
학 번	2018840814
이 름	**유지훈**
제출일	12/14/18

연세대학교

사전 조사 보고서

Page Global Directory, Page Upper Directory, Page Middle Directory, Page Table Entry 의 관계를 Linear Address 와 Physical Address 의 관계와 설명



리눅스에서 Page Global Directory, Page Upper Directory, Page Middle Directory, Page Table Entry 는 페이지 테이블 계층구조입니다. 리눅스에서 다양한 하드웨어를 지원또는 호환하기 위해 다음과 같이 4 단계로 페이지 테이블이 구성되었습니다. 기본적으로 cr4 레지스터는 page global directory 의 시작주소를 갖고 있고 Page global directory 는 Page Upper directory 의 시작 주소를, Page upper directory 는 Page middle directory 의 시작 주소를, Page middle directory 는 Page table 의 시작 주소를, page table 은 page 프레임의 물리주소를 갖고 있습니다. 각 해당 시작 주소에 offset 값을 더하여 가상주소의 실질적 물리주소를 찾을 수 있습니다. 즉, 그림 상단의 Linear Address 는 다음의 PGE, PUD, PMD, PTE 를 통해 Physical address 의 값을 구할 수 있습니다.

```
22   */
23  typedef struct { unsigned long  pte;    } pte_t;
24  typedef struct { unsigned long  ste[64];} pmd_t;
25  typedef struct { pmd_t          pue[1]; } pud_t;
26  typedef struct { pud_t          pge[1]; } pgd_t;
27  typedef struct { unsigned long  pgprot; } pgprot_t;
28  typedef struct page *pgtable_t;
29
```

각 계층구조의 엔트리 즉 시작을 나타내는 변수의 pgd_t, pud_t, pmd_t, pte_t 가 있습니다. 그리고 다음의 해당 frame 의 descriptor 를 구하는 방법에는 다음의 offset 매크로가 사용됩니다.

```
233
234  /* to find an entry in a page-table-directory */
235  static inline pgd_t * pgd_offset(const struct mm_struct *mm, unsigned long address)
236  {
237          return mm->pgd + pgd_index(address);
238  }
239
```

Pgd_t 값을 반환하는 매크로입니다.

```
502  /* Find an entry in the frst-level page table. */
503  #define pud_index(addr)         (((addr) >> PUD_SHIFT) & (PTRS_PER_PUD - 1))
504
505  static inline pud_t *pud_offset(pgd_t *pgd, unsigned long addr)
506  {
507          return (pud_t *)pgd_page_vaddr(*pgd) + pud_index(addr);
508  }
```

Pud_t 값을 반환하는 매크로입니다.

```
466  /* Find an entry in the second-level page table. */
467  #define pmd_index(addr)         (((addr) >> PMD_SHIFT) & (PTRS_PER_PMD - 1))
468
469  static inline pmd_t *pmd_offset(pud_t *pud, unsigned long addr)
470  {
471          return (pmd_t *)pud_page_vaddr(*pud) + pmd_index(addr);
472  }
```

Pmd_t 값을 반환하는 매크로입니다.

```
305  #define pte_offset(dir, address)
306          ((pte_t *) pmd_page_vaddr(*(dir)) + pte_index(address))
307  #define pte_offset_kernel(dir, address)
308          ((pte_t *) pmd_page_vaddr(*(dir)) + pte_index(address))
309  #define pte_offset_map(dir, address) pte_offset_kernel(dir, address)
310  #define pte_unmap(pte)          do { } while (0)
311
```

Pte_t 값을 반환하는 매크로입니다.

이들에 관계된 자료구조, 매크로, 함수, 매커니즘 흐름등을 항목별로 나눠서 process 별로 할당된 page 정보를 이용하여 물리 메모리를 참조하는 과정을 조사한다. (4.4.21)

자료구조(4.4.21)

/ include / linux / sched.h

```
1433
1434    #ifdef CONFIG_SCHED_INFO
1435            struct sched_info sched_info;
1436    #endif
1437
1438            struct list_head tasks;
1439    #ifdef CONFIG_SMP
1440            struct plist_node pushable_tasks;
1441            struct rb_node pushable_dl_tasks;
1442    #endif
1443
1444            struct mm_struct *mm, *active_mm;
1445            /* per-thread vma caching */
1446            u32 vmacache_seqnum;
1447            struct vm_area_struct *vmacache[VMACACHE_SIZE];
1448    #if defined(SPLIT_RSS_COUNTING)
1449            struct task_rss_stat    rss_stat;
1450    #endif
1451    /* task state */
```

다음은 task_struct 안에 mm_struct 를 가르키는 mm*입니다. 이 struct 포인터는 mm_struct 의 주소를 가르킵니다.

/ include / linux / mm_types.h

```
391  struct kioctx_table;
392  struct mm_struct {
393          struct vm_area_struct *mmap;            /* list of VMAs */
394          struct rb_root mm_rb;
395          u32 vmacache_seqnum;                   /* per-thread vmacache */
396  #ifdef CONFIG_MMU
397          unsigned long (*get_unmapped_area) (struct file *filp,
398                                  unsigned long addr, unsigned long len,
399                                  unsigned long pgoff, unsigned long flags);
400  #endif
401          unsigned long mmap_base;                /* base of mmap area */
402          unsigned long mmap_legacy_base;         /* base of mmap area in bottom-up allocations */
403          unsigned long task_size;                /* size of task vm space */
404          unsigned long highest_vm_end;           /* highest vma end address */
405          pgd_t * pgd;
406          atomic_t mm_users;                      /* How many users with user space? */
407          atomic_t mm_count;                      /* How many references to "struct mm_struct" (users count as 1) */
408          atomic_long_t nr_ptes;                  /* PTE page table pages */
409  #if CONFIG_PGTABLE_LEVELS > 2
410          atomic_long_t nr_pmds;                  /* PMD page table pages */
411  #endif
412          int map_count;                          /* number of VMAs */
413
414          spinlock_t page_table_lock;             /* Protects page tables and some counters */
415          struct rw_semaphore mmap_sem;
416
417          struct list_head mmlist;                /* List of maybe swapped mm's.  These are globally strung
418                                                   * together off init_mm.mmlist, and are protected
419                                                   * by mmlist_lock
420                                                   */
421
```

다음은 mm_struct 의 실질적 struct 를 나타내는 코드입니다.

/ include / linux / mm_types.h

```
391  struct kioctx_table;
392  struct mm_struct {
393          struct vm_area_struct *mmap;            /* list of VMAs */
394          struct rb_root mm_rb;
395          u32 vmacache_seqnum;                   /* per-thread vmacache */
396  #ifdef CONFIG_MMU
397          unsigned long (*get_unmapped_area) (struct file *filp,
398                                  unsigned long addr, unsigned long len,
399                                  unsigned long pgoff, unsigned long flags);
400  #endif
401          unsigned long mmap_base;                /* base of mmap area */
402          unsigned long mmap_legacy_base;         /* base of mmap area in bottom-up allocations */
403          unsigned long task_size;                /* size of task vm space */
404          unsigned long highest_vm_end;           /* highest vma end address */
405          pgd_t * pgd;
406          atomic_t mm_users;                      /* How many users with user space? */
407          atomic_t mm_count;                      /* How many references to "struct mm_struct" (users count as 1) */
408          atomic_long_t nr_ptes;                  /* PTE page table pages */
409  #if CONFIG_PGTABLE_LEVELS > 2
410          atomic_long_t nr_pmds;                  /* PMD page table pages */
411  #endif
```

다음은 mm_struct 안의 page global directory 를 가르키는 pgd_t*입니다.

```
22   */
23  typedef struct { unsigned long  pte;    } pte_t;
24  typedef struct { unsigned long  ste[64];} pmd_t;
25  typedef struct { pmd_t          pue[1]; } pud_t;
26  typedef struct { pud_t          pge[1]; } pgd_t;
27  typedef struct { unsigned long  pgprot; } pgprot_t;
28  typedef struct page *pgtable_t;
29
```

pte_t ,pmd_t, pgd_t, pgd_t 는 다음과 같이 정의됩니다.

```
29
30  /* PMD_SHIFT determines the size of the area a second-level page table can map */
31  #define PMD_SHIFT       (PAGE_SHIFT + (PAGE_SHIFT-3))
32  #define PMD_SIZE        (1UL << PMD_SHIFT)
33  #define PMD_MASK        (~(PMD_SIZE-1))
34
35  /* PGDIR_SHIFT determines what a third-level page table entry can map */
36  #define PGDIR_SHIFT     (PAGE_SHIFT + 2*(PAGE_SHIFT-3))
37  #define PGDIR_SIZE      (1UL << PGDIR_SHIFT)
38  #define PGDIR_MASK      (~(PGDIR_SIZE-1))
39
40  /*
41   * Entries per page directory level:  the Alpha is three-level, with
42   * all levels having a one-page page table.
43   */
44  #define PTRS_PER_PTE    (1UL << (PAGE_SHIFT-3))
45  #define PTRS_PER_PMD    (1UL << (PAGE_SHIFT-3))
46  #define PTRS_PER_PGD    (1UL << (PAGE_SHIFT-3))
47  #define USER_PTRS_PER_PGD       (TASK_SIZE / PGDIR_SIZE)
48  #define FIRST_USER_ADDRESS      0UL
49
```

Paging 에 관련하여 pre-defined 된 값들이 선언되어 있습니다. 이 정보들은 page table 의 주소 및 사이즈 등 페이지 테이블에 관련된 정보를 담고 있습니다. 다음의 정보를 통해, 좀 더 효과적으로 page table 에 대한 정보를 얻고 다룰 수 있습니다.

```
#define _PAGE_DIRTY     0x20000
#define _PAGE_ACCESSED  0x40000

/*
 * NOTE! The "accessed" bit isn't necessarily exact:  it can be kept exactly
 * by software (use the KRE/URE/KWE/UWE bits appropriately), but I'll fake it.
 * Under Linux/AXP, the "accessed" bit just means "read", and I'll just use
 * the KRE/URE bits to watch for it. That way we don't need to overload the
 * KWE/UWE bits with both handling dirty and accessed.
 *
 * Note that the kernel uses the accessed bit just to check whether to page
 * out a page or not, so it doesn't have to be exact anyway.
 */

#define __DIRTY_BITS    (_PAGE_DIRTY | _PAGE_KWE | _PAGE_UWE)
#define __ACCESS_BITS   (_PAGE_ACCESSED | _PAGE_KRE | _PAGE_URE)

#define _PFN_MASK       0xFFFFFFFF00000000UL

#define _PAGE_TABLE     (_PAGE_VALID | __DIRTY_BITS | __ACCESS_BITS)
#define _PAGE_CHG_MASK  (_PFN_MASK | __DIRTY_BITS | __ACCESS_BITS)
```

page 들의 bit 관련정보들이 선언되어 있습니다. 다음의 해당 테이블의 bit 상태를 파악하여 좀 더 페이지 테이블의 디테일한 정보를 얻을 수 있습니다.

```
01
02  /* Find an entry in the second-level page table.. */
03  extern inline pmd_t * pmd_offset(pgd_t * dir, unsigned long address)
04  {
05          pmd_t *ret = (pmd_t *) pgd_page_vaddr(*dir) + ((address >> PMD_SHIFT) & (PTRS_PER_PAGE - 1)
06          smp_read_barrier_depends(); /* see above */
07          return ret;
08  }
09
10  /* Find an entry in the third-level page table.. */
11  extern inline pte_t * pte_offset_kernel(pmd_t * dir, unsigned long address)
12  {
13          pte_t *ret = (pte_t *) pmd_page_vaddr(*dir)
14                  + ((address >> PAGE_SHIFT) & (PTRS_PER_PAGE - 1));
15          smp_read_barrier_depends(); /* see above */
16          return ret;
17  }
```

다음은 해당 page 의 단계별 offset 을 찾는 함수가 정의되어 있는 부분입니다. Architecture 의 bit 에 맞춰서 page level 을 정할 수 있습니다. 다음의 offset 매크로들을 통해 해당 페이지의 시작 주소점을 찾아 physical address 를 계산하여 얻을 수 있습니다.

매크로(4.4.21)

```
233
234  /* to find an entry in a page-table-directory */
235  static inline pgd_t * pgd_offset(const struct mm_struct *mm, unsigned long address)
236  {
237          return mm->pgd + pgd_index(address);
238  }
239
```

Pgd_t 값을 반환하는 매크로입니다.

Pgd 의 base 값에 pgd_offset 값을 더하여 해당 page global directory 의 값 및 page frame number 를 구합니다

```
502  /* Find an entry in the frst-level page table. */
503  #define pud_index(addr)         (((addr) >> PUD_SHIFT) & (PTRS_PER_PUD - 1))
504
505  static inline pud_t *pud_offset(pgd_t *pgd, unsigned long addr)
506  {
507          return (pud_t *)pgd_page_vaddr(*pgd) + pud_index(addr);
508  }
```

Pud_t 값을 반환하는 매크로입니다.

Pgd_offset 으로 구한 값에 pud_offset 값을 더하여 해당 page upper directory 의 값 및 page frame number 를 구합니다

```
466  /* Find an entry in the second-level page table. */
467  #define pmd_index(addr)         (((addr) >> PMD_SHIFT) & (PTRS_PER_PMD - 1))
468
469  static inline pmd_t *pmd_offset(pud_t *pud, unsigned long addr)
470  {
471          return (pmd_t *)pud_page_vaddr(*pud) + pmd_index(addr);
472  }
```

Pmd_t 값을 반환하는 매크로입니다.

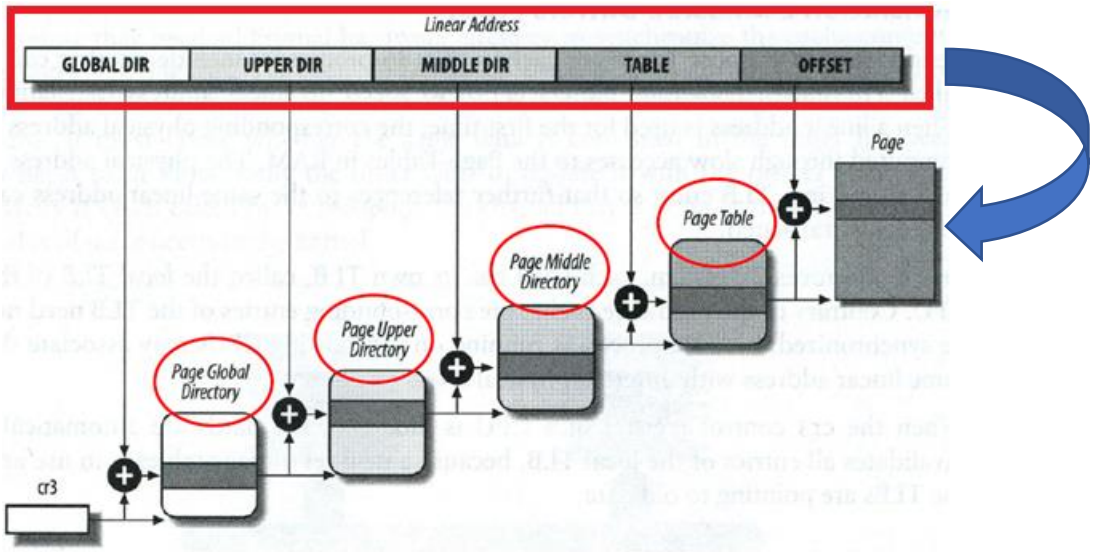
Pud_offset 으로 구한 값에 pmd_offset 값을 더하여 해당 page middle directory 의 값 및 page frame number 를 구합니다.

```
310  /* Find an entry in the third-level page table.. */
311  extern inline pte_t * pte_offset_kernel(pmd_t * dir, unsigned long address)
312  {
313          pte_t *ret = (pte_t *) pmd_page_vaddr(*dir)
314                  + ((address >> PAGE_SHIFT) & (PTRS_PER_PAGE - 1));
315          smp_read_barrier_depends(); /* see above */
316          return ret;
317  }
```

Pte_t 값을 반환하는 매크로입니다.

Pmd_offset 으로 구한 값에 ptd_offset 값을 더하여 해당 page table 의 값 및 page frame number 를 구합니다.

매커니즘과 함수들의 흐름(4.4.21)

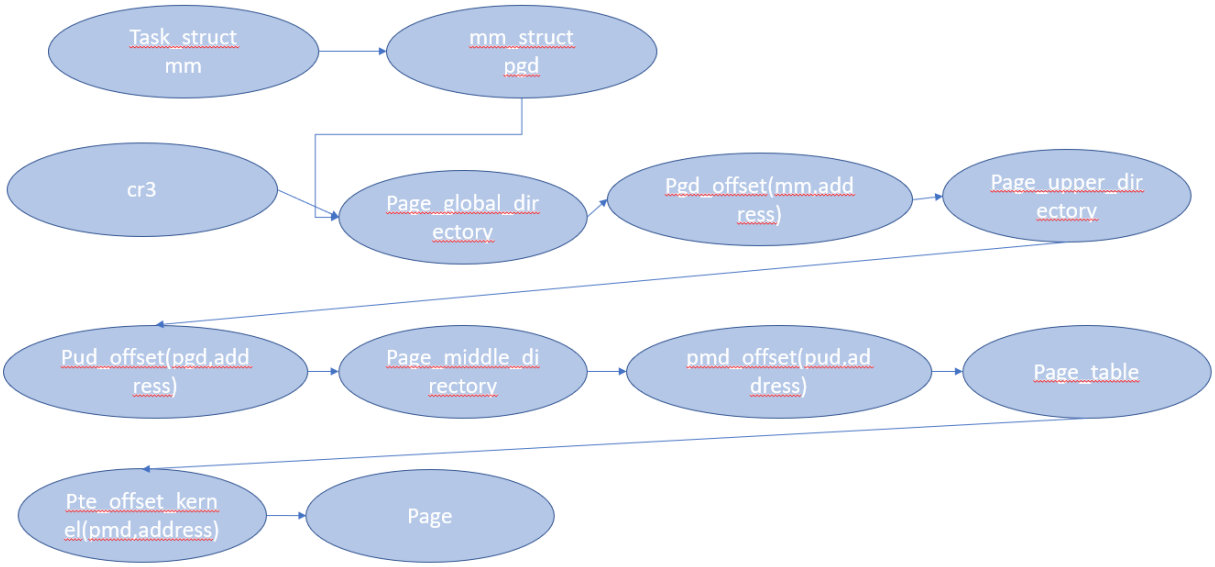


리눅스에서 Page Global Directory, Page Upper Directory, Page Middle Directory, Page Table Entry 는 페이지 테이블 계층구조입니다. 리눅스에서 다양한 하드웨어를 지원또는 호환하기 위해 다음과 같이 4 단계로 페이지 테이블이 구성되었습니다. 기본적으로 cr4 레지스터는 page global directory 의 시작주소를 갖고 있고 Page global directory 는 Page Upper directory 의 시작 주소를, Page upper direcotry 는 Page middle directory 의 시작 주소를, Page middle directory 는 Page table 의 시작 주소를, page table 은 page 프레임의 물리주소를 갖고 있습니다. 각 해당 시작 주소에 offset 값을 더하여 가상주소의 실질적 물리주소를 찾을 수 있습니다. 즉, 그림 상단의 Linear Address 는 다음의 PGE, PUD, PMD, PTE 를 통해 Physical address 의 값을 구할 수 있습니다.

```
/*
 * Change virtual addresses to physical addresses and vv.
 */
#ifdef USE_48_BIT_KSEG
static inline unsigned long virt_to_phys(void *address)
{
        return (unsigned long)address - IDENT_ADDR;
}

static inline void * phys_to_virt(unsigned long address)
{
        return (void *) (address + IDENT_ADDR);
}
#else
static inline unsigned long virt_to_phys(void *address)
{
        unsigned long phys = (unsigned long)address;

        /* Sign-extend from bit 41.  */
        phys <<= (64 - 41);
        phys = (long)phys >> (64 - 41);

        /* Crop to the physical address width of the processor.  */
        phys &= (1ul << hwrpb->pa_bits) - 1;

        return phys;
}

static inline void * phys_to_virt(unsigned long address)
{
        return (void *)(IDENT_ADDR + (address & ((1ul << 41) - 1)));
}
#endif

#define page_to_phys(page)      page_to_pa(page)
```

다음은 virtual address 를 physical address 로 바꾸는 함수 또는 physical address 를 virtual address 로 바꾸는 함수입니다.

순서도(4.4.21)


Task_struct
mm
mm_struct
pgd
cr3
Page_global_dir
ectory
Pgd_offset(mm,add
ress)
Page_upper_dir
ectory
Pud_offset(pgd,add
ress)
Page_middle_di
rectory
pmd_offset(pud,ad
dress)
Page_table
Pte_offset_kern
el(pmd,address)
Page

이들에 관계된 자료구조, 매크로, 함수, 매커니즘 흐름등을 항목별로 나눠서 process 별로 할당된 page 정보를 이용하여 물리 메모리를 참조하는 과정을 조사한다. (2.6.10)

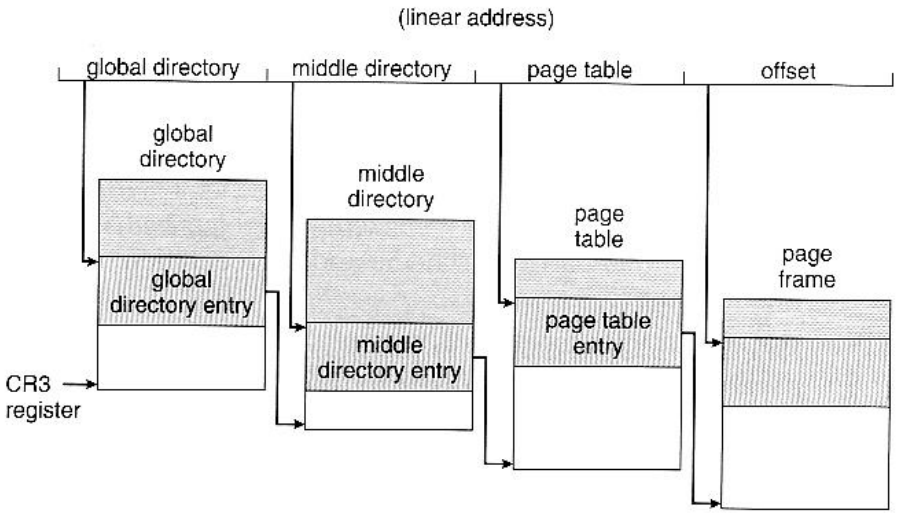


Figure 8.24 Three-level paging in Linux.

"Linux Kernel 은 2.6.10 버전 까지는 32bit 물리적 메모리 를 지원하기 위해서 3-level paging 기법이 적용되어 있습니다."

자료구조(2.6.10)

```
537  #endif
538
539      struct list_head tasks;
540      /*
541       * ptrace_list/ptrace_children forms the list of my children
542       * that were stolen by a ptracer.
543       */
544      struct list_head ptrace_children;
545      struct list_head ptrace_list;
546
547      struct mm_struct *mm, *active_mm;
548
```

다음은 task_struct 안의 mm_struct 포인터 mm 입니다.

```
204 struct mm_struct {
205     struct vm_area_struct * mmap;       /* list of VMAs */
206     struct rb_root mm_rb;
207     struct vm_area_struct * mmap_cache; /* last find_vma result */
208     unsigned long (*get_unmapped_area) (struct file *filp,
209                 unsigned long addr, unsigned long len,
210                 unsigned long pgoff, unsigned long flags);
211     void (*unmap_area) (struct vm_area_struct *area);
212     unsigned long mmap_base;        /* base of mmap area */
213     unsigned long free_area_cache;      /* first hole */
214     pgd_t * pgd;
215     atomic_t mm_users;          /* How many users with user space? */
216     atomic_t mm_count;          /* How many references to "struct mm_struct" (users count as 1) */
217     int map_count;              /* number of VMAs */
218     struct rw_semaphore mmap_sem;
219     spinlock_t page_table_lock;     /* Protects page tables, mm->rss, mm->anon_rss */
220
221     struct list_head mmlist;        /* List of maybe swapped mm's.  These are globally strung
222                                      * together off init_mm.mmlist, and are protected
223                                      * by mmlist_lock
224                                      */
225
226     unsigned long start_code, end_code, start_data, end_data;
227     unsigned long start_brk, brk, start_stack;
228     unsigned long arg_start, arg_end, env_start, env_end;
229     unsigned long rss, anon_rss, total_vm, locked_vm, shared_vm;
230     unsigned long exec_vm, stack_vm, reserved_vm, def_flags, nr_ptes;
231
232     unsigned long saved_auxv[42]; /* for /proc/PID/auxv */
233
234     unsigned dumpable:1;
235     cpumask_t cpu_vm_mask;
236
237     /* Architecture-specific MM context */
238     mm_context_t context;
239
240     /* Token based thrashing protection. */
241     unsigned long swap_token_time;
```

다음은 mm_struct 의 실질적 struct 구조입니다.

```
25 /*
26  * These are used to make use of C type-checking..
27  */
28 typedef struct { unsigned long pte; } pte_t;
29 typedef struct { unsigned long pmd; } pmd_t;
30 typedef struct { unsigned long pgd; } pgd_t;
31 typedef struct { unsigned long pgprot; } pgprot_t;
32
33 #define pte_val(x)  ((x).pte)
34 #define pmd_val(x)  ((x).pmd)
35 #define pgd_val(x)  ((x).pgd)
36 #define pgprot_val(x)   ((x).pgprot)
37
```

3 level page 기법이 사용되었으므로 실질적 커널 코드에 pud 를 제외한 세개의 pte,pmd,pgd 가 정의되어있습니다.

```
22  #define set_pte(pteptr, pteval) ((*(pteptr)) = (pteval))
23
24  /* PMD_SHIFT determines the size of the area a second-level page table can map */
25  #define PMD_SHIFT   (PAGE_SHIFT + (PAGE_SHIFT-3))
26  #define PMD_SIZE    (1UL << PMD_SHIFT)
27  #define PMD_MASK    (~(PMD_SIZE-1))
28
29  /* PGDIR_SHIFT determines what a third-level page table entry can map */
30  #define PGDIR_SHIFT (PAGE_SHIFT + 2*(PAGE_SHIFT-3))
31  #define PGDIR_SIZE  (1UL << PGDIR_SHIFT)
32  #define PGDIR_MASK  (~(PGDIR_SIZE-1))
33
```

각종 pre-defined 된 paging 에 관련한 값들이 선언되어 있습니다. 이 정보들은 page table 의 주소 및 사이즈 등 페이지 테이블에 관련된 정보를 담고 있습니다. 다음의 정보를 통해, 좀 더 효과적으로 page table 에 대한 정보를 얻고 다룰 수 있습니다.

```
67  /* .. and these are ours ... */
68  #define _PAGE_DIRTY 0x20000
69  #define _PAGE_ACCESSED  0x40000
70  #define _PAGE_FILE  0x80000 /* set:pagecache, unset:swap */
71
72  /*
73   * NOTE! The "accessed" bit isn't necessarily exact:  it can be kept exactly
74   * by software (use the KRE/URE/KWE/UWE bits appropriately), but I'll fake it.
75   * Under Linux/AXP, the "accessed" bit just means "read", and I'll just use
76   * the KRE/URE bits to watch for it. That way we don't need to overload the
77   * KWE/UWE bits with both handling dirty and accessed.
78   *
79   * Note that the kernel uses the accessed bit just to check whether to page
80   * out a page or not, so it doesn't have to be exact anyway.
81   */
82
83  #define __DIRTY_BITS    (_PAGE_DIRTY | _PAGE_KWE | _PAGE_UWE)
84  #define __ACCESS_BITS   (_PAGE_ACCESSED | _PAGE_KRE | _PAGE_URE)
85
86  #define _PFN_MASK   0xFFFFFFFF00000000UL
87
88  #define _PAGE_TABLE (_PAGE_VALID | __DIRTY_BITS | __ACCESS_BITS)
89  #define _PAGE_CHG_MASK  (_PFN_MASK | __DIRTY_BITS | __ACCESS_BITS)
90
```

Page 의 bit 관련 정보들이 있습니다. 다음의 해당 테이블의 bit 상태를 파악하여 좀 더 페이지 테이블의 디테일한 정보를 얻을 수 있습니다.

```
#define PAGE_DIR_OFFSET(tsk,address) pgd_offset((tsk),(address))

/* to find an entry in a kernel page-table-directory */
#define pgd_offset_k(address) pgd_offset(&init_mm, address)

/* to find an entry in a page-table-directory. */
#define pgd_index(address)  ((address >> PGDIR_SHIFT) & (PTRS_PER_PGD - 1))
#define pgd_offset(mm, address) ((mm)->pgd+pgd_index(address))

/* Find an entry in the second-level page table.. */
extern inline pmd_t * pmd_offset(pgd_t * dir, unsigned long address)
{
    return (pmd_t *) pgd_page(*dir) + ((address >> PMD_SHIFT) & (PTRS_PER_PAGE - 1));
}

/* Find an entry in the third-level page table.. */
extern inline pte_t * pte_offset_kernel(pmd_t * dir, unsigned long address)
{
    return (pte_t *) pmd_page_kernel(*dir)
        + ((address >> PAGE_SHIFT) & (PTRS_PER_PAGE - 1));
}
```

다음은 해당 page 의 단계별 offset 을 찾는 함수가 정의되어 있는 부분입니다. Architecture 의 bit 에 맞춰서 page level 을 정할 수 있습니다. 다음의 offset 매크로들을 통해 해당 페이지의 시작 주소점을 찾아 physical address 를 계산하여 얻을 수 있습니다.

매크로(2.6.10)

```
275  /* to find an entry in a page-table-directory. */
276  #define pgd_index(address)  ((address >> PGDIR_SHIFT) & (PTRS_PER_PGD - 1))
277  #define pgd_offset(mm, address) ((mm)->pgd+pgd_index(address))
```

Pgd_t 값을 반환하는 매크로입니다.

Pgd 의 base 값에 pgd_offset 값을 더하여 해당 page global directory 의 값 및 page frame number 를 구합니다

```
279  /* Find an entry in the second-level page table.. */
280  extern inline pmd_t * pmd_offset(pgd_t * dir, unsigned long address)
281  {
282      return (pmd_t *) pgd_page(*dir) + ((address >> PMD_SHIFT) & (PTRS_PER_PAGE - 1));
283  }
284
```

Pmd_t 값을 반환하는 매크로입니다.

Pgd_offset 으로 구한 값에 pmd_offset 값을 더하여 해당 page middle directory 의 값 및 page frame number 를 구합니다

```
285  /* Find an entry in the third-level page table.. */
286  extern inline pte_t * pte_offset_kernel(pmd_t * dir, unsigned long address)
287  {
288      return (pte_t *) pmd_page_kernel(*dir)
289          + ((address >> PAGE_SHIFT) & (PTRS_PER_PAGE - 1));
290  }
291
```

Pte_t 값을 반환하는 매크로입니다.

Pmd_offset 으로 구한 값에 ptd_offset 값을 더하여 해당 page table 의 값 및 page frame number 를 구합니다.

매커니즘과 함수들의 흐름(2.6.10)

리눅스(커널 2.6.10)에서 Page Global Directory, Page Middle Directory, Page Table Entry 는 페이지 테이블 계층구조입니다. 리눅스에서 다양한 하드웨어를 지원또는 호환하기 위해 다음과 같이 3 단계로 페이지 테이블이 구성되었습니다. 기본적으로 cr4 레지스터는 page global directory 의 시작주소를 갖고 있고 Page global directory 는 Page Middle directory 의 시작 주소를, Page middle directory 는 Page table 의 시작 주소를, page table 은 page 프레임의 물리주소를 갖고 있습니다. 각 해당 시작 주소에 offset 값을 더하여 가상주소의 실질적 물리주소를 찾을 수 있습니다. 즉, 그림 상단의 Linear Address 는 다음의 PGE, PMD, PTE 를 통해 Physical address 의 값을 구할 수 있습니다.

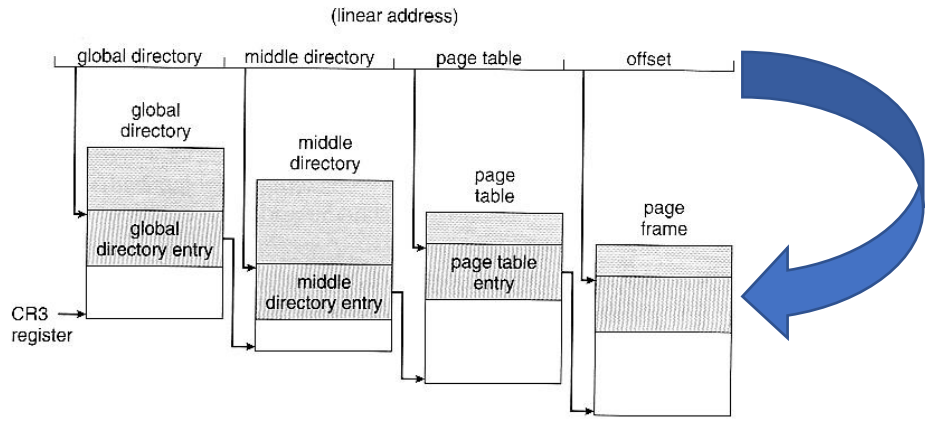
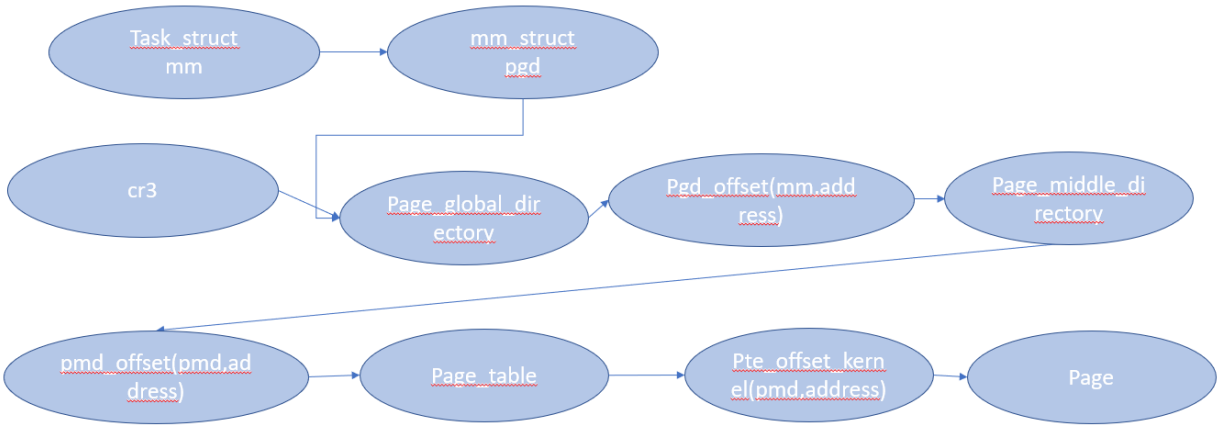


Figure 8.24 Three-level paging in Linux.

```
58  /*
59   * Change virtual addresses to physical addresses and vv.
60   */
61  #ifdef USE_48_BIT_KSEG
62  static inline unsigned long virt_to_phys(void *address)
63  {
64      return (unsigned long)address - IDENT_ADDR;
65  }
66
67  static inline void * phys_to_virt(unsigned long address)
68  {
69      return (void *) (address + IDENT_ADDR);
70  }
71  #else
72  static inline unsigned long virt_to_phys(void *address)
73  {
74          unsigned long phys = (unsigned long)address;
75
76      /* Sign-extend from bit 41.  */
77      phys <<= (64 - 41);
78      phys = (long)phys >> (64 - 41);
79
80      /* Crop to the physical address width of the processor.  */
81          phys &= (1ul << hwrpb->pa_bits) - 1;
82
83          return phys;
84  }
85
86  static inline void * phys_to_virt(unsigned long address)
87  {
88          return (void *)(IDENT_ADDR + (address & ((1ul << 41) - 1)));
89  }
90  #endif
```

다음은 virtual address 를 physical address 로 바꾸는 함수 또는 physical address 를 virtual address 로 바꾸는 함수입니다.

순서도(2.6.10)


Task_struct
mm
mm_struct
pgd
cr3
Page_global_dir
ectory
Pgd_offset(mm,add
ress)
Page_middle_di
rectory
pmd_offset(pmd,ad
dress)
Page_table
Pte_offset_kern
el(pmd,address)
Page

3-level paging 과 4-level paging 비교 분석

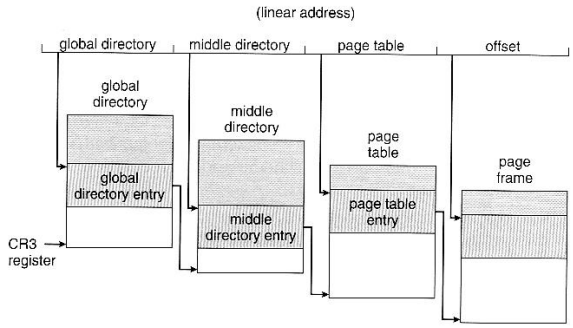
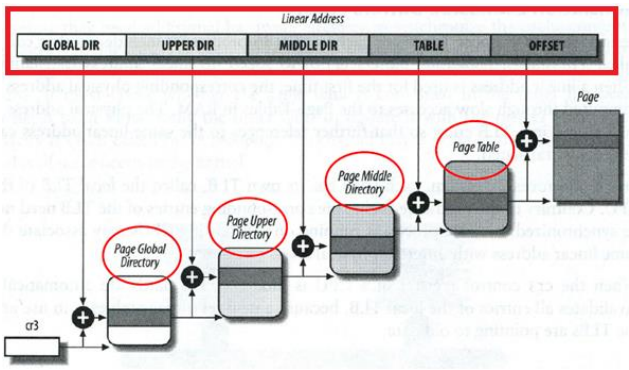


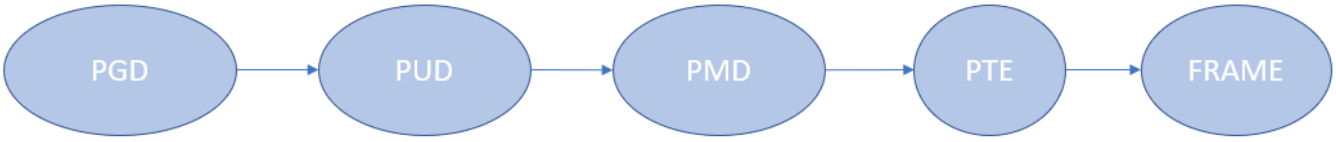
Figure 8.24 Three-level paging in Linux.



다음은 3-level paging 과 4-level paging 을 나타내는 사진입니다. 3-level paging 은 page global directory, page middle directory, page table 그리고 page 로 구성됩니다. 반면에 4-level paging 은 page global directory, page upper directory, page middle directory, page table 로 구성됩니다.

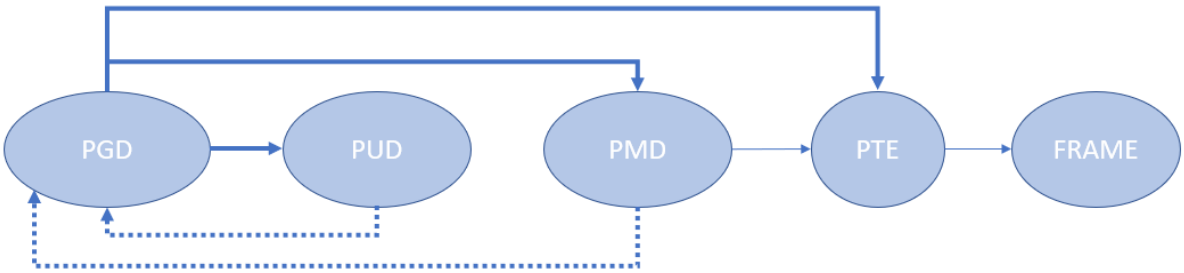
64bit architecture 에서는 3-level paging 을 씁니다. 하지만, 리눅스에서는 4-level paging 을 지원합니다. 리눅스에서 4-level paging 을 가능하도록 만들어 놓은 이유는 나중에 더 bit 가 많은 architecture 가 나올 수 있기 때문입니다(primary goal of linux is portability). 4-level paging 을 구현해 놓음으로써 리눅스는 targeting architecture 에 따라 configure 하여 paging 을 사용할 수 있습니다.

3-level page 에서의 page frame 은 8kb 고 offset filed 는 13bits 입니다. 43 least significant bits 가 주소로 사용되므로, 2^43 에서 8TB address space 를 갖습니다.



실질적 4-level page 는 다음과 같습니다.

4-level page 의 경우 좀 더 많은 bit 의 architecture 를 호환할 수 있습니다. 또한, 좀 더 적은 bit 의 architecture 를 호환할 경우 (two-level paging hardware), Page Upper Directory 와 Page Middle Directory filed 를 zero bits 로 configure 해서 two-level paging 으로 사용할 수 있습니다.



4-level page 로 2-level page 를 구현하면 다음과 같이 작동합니다.

사용자 프로그램에서 일정 크기의 메모리를 할당 했다가 해제 할 시 커널 내부에서 일어나는 동작을 설명

메모리를 해체할 시 커널 내부에서 일어나는 동작은 mm/page_alloc.c 에서 확인할 수 있습니다.
또한 함수를 분석하여 보면 buddy system 을 통해 메모리 할당을 해체 하는 과정을 볼 수 있습니다.

```
3317  void free_pages(unsigned long addr, unsigned int order)
3318  {
3319          if (addr != 0) {
3320                  VM_BUG_ON(!virt_addr_valid((void *)addr));
3321                  __free_pages(virt_to_page((void *)addr), order);
3322          }
3323  }
3324
```

Free_pages 에서 addr 값과 order 값을 받는다. addr 값이 0 이아니면 __free_pages 를 진행한다.
또한 해당 virtual address 를 page 로 바꾸는 함수를 사용하여 __free pages 를 실행한다.

```
3305  void __free_pages(struct page *page, unsigned int order)
3306  {
3307          if (put_page_testzero(page)) {
3308                  if (order == 0)
3309                          free_hot_cold_page(page, false);
3310                  else
3311                          __free_pages_ok(page, order);
3312          }
3313  }
```

__free_pages 에서는 order 값이 0 이면 free_hot_colde_page 를 실행하고, 그 외에 값은
__free_pages_ok 를 실행한다. 여기서 hot, cold 는 head 와 tail 의 위치에 대응하여 관리됩니다.
Hot 은 리스트의 앞부분에 페이지들과 allocate 될 가능성이 높고, cold 는 뒷 부분의 리스트에 있는 페이지와 통합될 가능성이 높은 것 입니다..

```
1011
1012  static void __free_pages_ok(struct page *page, unsigned int order)
1013  {
1014          unsigned long flags;
1015          int migratetype;
1016          unsigned long pfn = page_to_pfn(page);
1017
1018          if (!free_pages_prepare(page, order))
1019                  return;
1020
1021          migratetype = get_pfnblock_migratetype(page, pfn);
1022          local_irq_save(flags);
1023          __count_vm_events(PGFREE, 1 << order);
1024          free_one_page(page_zone(page), page, pfn, order, migratetype);
1025          local_irq_restore(flags);
1026  }
1027
```

```
855  static void free_one_page(struct zone *zone,
856                                struct page *page, unsigned long pfn,
857                                unsigned int order,
858                                int migratetype)
859  {
860          unsigned long nr_scanned;
861          spin_lock(&zone->lock);
862          nr_scanned = zone_page_state(zone, NR_PAGES_SCANNED);
863          if (nr_scanned)
864                  __mod_zone_page_state(zone, NR_PAGES_SCANNED, -nr_scanned);
865
866          if (unlikely(has_isolate_pageblock(zone) ||
867                  is_migrate_isolate(migratetype))) {
868                  migratetype = get_pfnblock_migratetype(page, pfn);
869          }
870          __free_one_page(page, pfn, zone, order, migratetype);
871          spin_unlock(&zone->lock);
872  }
```

Free_one_page 의 경우, 함수 실행 전에 spin_lock 을 통해 해당 data 를 보호하고 __free_one_page 를 실행합니다.

```
657
658  static inline void free_one_page(struct page *page,
659                  unsigned long pfn,
660                  struct zone *zone, unsigned int order,
661                  int migratetype)
662  {
663          unsigned long page_idx;
664          unsigned long combined_idx;
665          unsigned long uninitialized_var(buddy_idx);
666          struct page *buddy;
667          unsigned int max_order;
668
669          max_order = min_t(unsigned int, MAX_ORDER, pageblock_order + 1);
670
671          VM_BUG_ON(!zone_is_initialized(zone));
672          VM_BUG_ON_PAGE(page->flags & PAGE_FLAGS_CHECK_AT_PREP, page);
673
674          VM_BUG_ON(migratetype == -1);
675          if (likely(!is_migrate_isolate(migratetype)))
676                  __mod_zone_freepage_state(zone, 1 << order, migratetype);
677
678          page_idx = pfn & ((1 << MAX_ORDER) - 1);
679
680          VM_BUG_ON_PAGE(page_idx & ((1 << order) - 1), page);
681          VM_BUG_ON_PAGE(bad_range(zone, page), page);
682
```

__free_one_page 에서 실질적으로 page 가 합쳐지는 아닌지에 대한 판별이 buddy system 을 실행하는 알고리즘에 따라 실행되고 정해집니다.

해당 부분의 Buddy Memory Allocator 는 메모리 allocation 관리를 위한 시스템으로 free 한 연속된 페이지의 해지를 관리하며 최대한 fragmentation 이 되지 않도록 합니다. 1 페이지부터

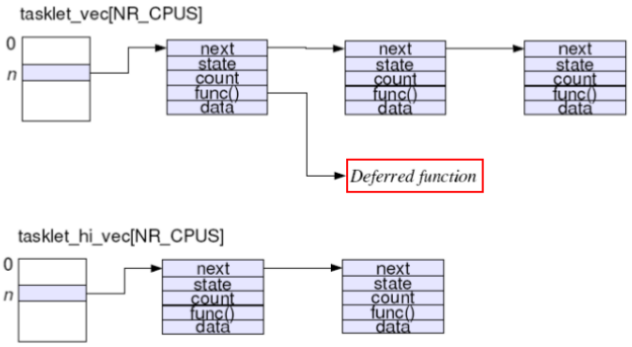
해당 코드에서 정의된 2 의 MAX_ORDER-1 값을 최대값으로 하여 관리됩니다.

인터럽트 지연 처리를 위한 tasklet 자료구조에 대해서 분석 + 수행되는 매커니즘 분석 설명

Tasklet 은 인터럽트 지연처리(bottom half)를 위한 것으로 softirq 메카니즘을 통해 수행된다.

```
struct tasklet_struct
{
        struct tasklet_struct *next;
        unsigned long state;
        atomic_t count;
        void (*func)(unsigned long);
        unsigned long data;
};
```

다음은 tasklet_struct 의 자료구조입니다. *next 의 경우 리스트의 다음 tasklet 을 가르킵니다. Unsigned long State 는 tasklet 의 상태를 말하고, count 는 카운터입니다. Void((func)(unsigned long)은 tasklet handler function 입니다.(deferred function) Data 는 tasklet function 의 argument 입니다.



해당 tasklet_struct 의 구조는 수업시간에 배운 다음의 tasklet 자료 구조와 일치합니다.

```
static DEFINE_PER_CPU(struct tasklet_head, tasklet_vec);
static DEFINE_PER_CPU(struct tasklet_head, tasklet_hi_vec);
```

다음은 CPU 마다 있는 두 개의 tasklet 리스트 입니다. (kernel/softirq.c)

```
#define DECLARE_TASKLET(name, func, data) \
struct tasklet_struct name = { NULL, 0, ATOMIC_INIT(0), func, data }

#define DECLARE_TASKLET_DISABLED(name, func, data) \
struct tasklet_struct name = { NULL, 0, ATOMIC_INIT(1), func, data }
```

tasklet 을 선언하는 것 입니다. (interrupt.h)

```
extern void __tasklet_schedule(struct tasklet_struct *t);

static inline void tasklet_schedule(struct tasklet_struct *t)
{
        if (!test_and_set_bit(TASKLET_STATE_SCHED, &t->state))
                __tasklet_schedule(t);
}

extern void __tasklet_hi_schedule(struct tasklet_struct *t);

static inline void tasklet_hi_schedule(struct tasklet_struct *t)
{
        if (!test_and_set_bit(TASKLET_STATE_SCHED, &t->state))
                __tasklet_hi_schedule(t);
}
```

두 함수는 __tasklet_schedule(t) or __tasklet_hi_schedule(t)를 호출하면 다음의 코드를 진행합니다.

```
void __tasklet_schedule(struct tasklet_struct *t)
{
        unsigned long flags;

        local_irq_save(flags);
        t->next = NULL;
        *__this_cpu_read(tasklet_vec.tail) = t;
        __this_cpu_write(tasklet_vec.tail, &(t->next));
        raise_softirq_irqoff(TASKLET_SOFTIRQ);
        local_irq_restore(flags);
}
EXPORT_SYMBOL(__tasklet_schedule);

void __tasklet_hi_schedule(struct tasklet_struct *t)
{
        unsigned long flags;

        local_irq_save(flags);
        t->next = NULL;
        *__this_cpu_read(tasklet_hi_vec.tail) = t;
        __this_cpu_write(tasklet_hi_vec.tail,  &(t->next));
        raise_softirq_irqoff(HI_SOFTIRQ);
        local_irq_restore(flags);
}
EXPORT_SYMBOL(__tasklet_hi_schedule);
```

위에서 호출한 tasklet_schedule 로 해당 tasklet 을 tasklet list 에 add 하고 softirq 를 raise 합니다. Local_irq_save 는 현재 local interrupt delivery 상태를 저장하고 disable 합니다. Local_irq_restore 은 local interrupt delivery 를 기존의 state 로 변환합니다.

```
485  static void tasklet_action(struct softirq_action *a)
486  {
487          struct tasklet_struct *list;
488
489          local_irq_disable();
490          list = __this_cpu_read(tasklet_vec.head);
491          __this_cpu_write(tasklet_vec.head, NULL);
492          __this_cpu_write(tasklet_vec.tail, this_cpu_ptr(&tasklet_vec.head)
493          local_irq_enable();
494
495          while (list) {
496                  struct tasklet_struct *t = list;
497
498                  list = list->next;
499
500                  if (tasklet_trylock(t)) {
501                          if (!atomic_read(&t->count)) {
502                                  if (!test_and_clear_bit(TASKLET_STATE_SCHE
503                                                          &t->state))
504                                          BUG();
505                                  t->func(t->data);
506                                  tasklet_unlock(t);
507                                  continue;
508                          }
509                          tasklet_unlock(t);
510                  }
511
512                  local_irq_disable();
513                  t->next = NULL;
514                  *__this_cpu_read(tasklet_vec.tail) = t;
515                  __this_cpu_write(tasklet_vec.tail, &(t->next));
516                  __raise_softirq_irqoff(TASKLET_SOFTIRQ);
517                  local_irq_enable();
518          }
519  }
```

void tasklet action 코드는 스케줄된 tasklet 을 실행하는 코드입니다. 하지만 실질적으로 이 tasklet_action 은 softirq 를 통해 수행됩니다. 이 코드의 t->func(t->data) 해당 tasklet 을 실행합니다. Local_irq_disable 은 interrupts 가 disabled 된 것 입니다. Local_irq_enalbe 은 다시 interrupts 가 enable 된 것입니다.

```
void __init softirq_init(void)
{
        int cpu;

        for_each_possible_cpu(cpu) {
                per_cpu(tasklet_vec, cpu).tail =
                        &per_cpu(tasklet_vec, cpu).head;
                per_cpu(tasklet_hi_vec, cpu).tail =
                        &per_cpu(tasklet_hi_vec, cpu).head;
        }

        open_softirq(TASKLET_SOFTIRQ, tasklet_action);
        open_softirq(HI_SOFTIRQ, tasklet_hi_action);
}
```

Softirq_init 에서 open_softirq 할 때, tasklet_action 또는 tasklet_hi_actino 을 호출한다.

```
void open_softirq(int nr, void (*action)(struct softirq_action *))
{
        softirq_vec[nr].action = action;
}
```

Open_softirq 는 해당 irq 에 해당하는 void(*action 을)갖고 struct softirq_actino *을 인자로 받고 actino 을 레지스터한다.

```
static inline void invoke_softirq(void)
{
        if (!force_irqthreads) {
#ifdef CONFIG_HAVE_IRQ_EXIT_ON_IRQ_STACK
                /*
                 * We can safely execute softirq on the current stack if
                 * it is the irq stack, because it should be near empty
                 * at this stage.
                 */
                __do_softirq();
#else
                /*
                 * Otherwise, irq_exit() is called on the task stack that can
                 * be potentially deep already. So call softirq in its own stack
                 * to prevent from any overrun.
                 */
                do_softirq_own_stack();
#endif
        } else {
                wakeup_softirqd();
        }
}
```

Invoke softirq 에서 do_softirq 를 호출합니다.

```
230  asmlinkage __visible void __do_softirq(void)
231  {
232          unsigned long end = jiffies + MAX_SOFTIRQ_TIME;
233          unsigned long old_flags = current->flags;
234          int max_restart = MAX_SOFTIRQ_RESTART;
235          struct softirq_action *h;
236          bool in_hardirq;
237          __u32 pending;
238          int softirq_bit;
239
240          /*
241           * Mask out PF_MEMALLOC s current task context is borrowed for the
242           * softirq. A softirq handled such as network RX might set PF_MEMALLOC
243           * again if the socket is related to swap
244           */
245          current->flags &= ~PF_MEMALLOC;
246
247          pending = local_softirq_pending();
248          account_irq_enter_time(current);
249
250          __local_bh_disable_ip(_RET_IP_, SOFTIRQ_OFFSET);
251          in_hardirq = lockdep_softirq_start();
252
253  restart:
254          /* Reset the pending bitmask before enabling irqs */
255          set_softirq_pending(0);
256
257          local_irq_enable();
258
```

다음의 do_softirq function 은 말 그대로 softirq 를 수행하는 것입니다. Cpu 에 Softirq 가 pending 인지 확인하고 처리합니다. 후에 wakeup_softirq()를 실행합니다.

```
 */
static void wakeup_softirqd(void)
{
        /* Interrupts are disabled: no need to stop preemption */
        struct task_struct *tsk = __this_cpu_read(ksoftirqd);

        if (tsk && tsk->state != TASK_RUNNING)
                wake_up_process(tsk);
}
```

Wakeup_softirq 는 ksoftirqd 를 실행합니다.

```
static int ksoftirqd_should_run(unsigned int cpu)
{
        return local_softirq_pending();
}

static void run_ksoftirqd(unsigned int cpu)
{
        local_irq_disable();
        if (local_softirq_pending()) {
                /*
                 * We can safely run softirq on inline stack, as we are not deep
                 * in the task stack here.
                 */
                __do_softirq();
                local_irq_enable();
                cond_resched_rcu_qs();
                return;
        }
        local_irq_enable();
}
```

실습 과제 보고서

프로그래밍 과제 수행을 위해 사용한 kernel 의 기능과 각 기능을 사용하기 위해 필요한 kernel 도구들을 요약한다.

#include <linux/random.h>

- 난수를 생성하는 함수에 쓰인 get_random_bytes 를 사용하기 위해 사용되었습니다.

#include <linux/sched.h>

- Tasklet struct 을 선언하기 위해 사용되었습니다.

#include <linux/moduleparam.h>

- 커멘드 창에서 period 주기 값을 얻기 위해 사용되었습니다.

#include <linux/random.h>

- 난수를 생성하는 get_random_int()를 사용하기 위해 선언되었습니다.

#include <linux/interrupt.h>

- Tasklet 을 사용하기 위해 선언되었습니다.

#include <asm/pgtable.h>

- Paging 정보를 얻기 위해 선언되었습니다.

작성한 프로그램의 소스코드 필요한 부분만 작성

```
1   #include <linux/kernel.h>
2   #include <linux/module.h>
3   #include <linux/init.h>
4   #include <linux/proc_fs.h>
5   #include <linux/random.h>
6   #include <linux/seq_file.h>
7   #include <linux/sched.h>
8   #include <linux/interrupt.h>
9   #include <linux/moduleparam.h>
10  #include <asm/pgtable.h>
11  #define PROC_NAME "hw2"
12  #define STUDENT_NAME "Jihon You"
13  #define STUDENT_ID "2018123123"
```

가장 상위에 있는 코드로 필요한 linux 의 부분을 참조하여 #include 합니다.

```
14
15  int period = 0;
16  module_param(period, int, 0);
17  #define placeholder 0
```

다음은 module parameter 를 사용하기 위해 선언된 것 입니다. Period 는 insmod 시 입력되는 숫자 값을 받아옵니다.

```
25  static void *my_seq_start(struct seq_file *s, loff_t *pos)
26  {
27  static unsigned long counter = 0;
28      /* beginning a new sequence ? */
29      if ( *pos == 0 )
30      {
31      /* yes => return a non null value to begin the sequence */
32      return &counter;
33      }
34      else
35      {
36      /* no => it's the end of the sequence, return end to stop reading */
37      *pos = 0;
38      return NULL;
39      }
40  }
41  /**
```

다음의 코드는 sequence 에서 가장 먼저 호출되는 함수입니다.

```
46  static void *my_seq_next(struct seq_file *s, void *v, loff_t *pos)
47  {
48      unsigned long *tmp_v = (unsigned long *)v;
49      (*tmp_v)++;
50      (*pos)++;
51      return NULL;
52  }
```

다음은 my_seq_start 이후에 불려지는 함수이며, next 가 NULL 이 될 때 까지 호출됩니다.

```
53  /**
54  * This function is called at the end of a sequence
55  *
56  */
57  static void my_seq_stop(struct seq_file *s, void *v)
58  {
59  /* nothing to do, we use a static value in start() */
60  }
61
```

다음은 sequence 의 마지막에 호출되는 함수이지만, 편의상 만들어 놓았습니다.

```
62  static void print_bar(struct seq_file *s) {
63      seq_printf(s, "************************************************************************\n");
64  }
```

다음은 바를 print 하는 함수입니다.

```
66  //check process
67  static int get_total_process(struct task_struct *task) {
68      int counter = 0;
69      for_each_process(task) {
70          //task->mm == null means kernel
71          if(task->mm != NULL){
72              counter++;
73          }
74      }
75      return counter;
76  }
```

이 함수는 tasklet 에서 사용되는 함수로, 쓰레드를 제외한 프로세스의 전체수를 구하는 함수입니다. Task->mm 이 thread 일 경우 null 값을 return 합니다. 그러므로 null 값이 아닌 것을 count 하면 process 의 총 수가 나옵니다.

```
81  static struct task_struct *check;
82  static struct task_struct *picked;
83
84  static void func(unsigned long data);
85  //declare tasklet
86  DECLARE_TASKLET(name, func, placeholder);
87  //tasklet to find process
88   static void func(unsigned long data){
89      //Current is a global variable of type struct task_struct
90      check = current;
91      int totalProcess = get_total_process(check);
92
93      int counter = 0;
94      int random = 0;
95      //get random number in bound of total process
96      random = get_random_int();
97
98      random = random%totalProcess;
99
100     for_each_process(check) {
101         //task->mm == null means kernel
102         if(check->mm != NULL) {
103             counter++;
104             if(counter == random) {
105                 picked = check;
106             }
107         }
108     }
109
110 }
```

다음은 tasklet 에서 실행되는 함수로, 과제 스펙에서와 같이 랜덤한 process 를 선택합니다. TotalProcess 는 위에 설명한 함수에서 구해지고, 난수는 get_random_int()로 구해집니다. 난수의 값이 process 총 수보다 커지는 것을 대비해 totalprocess 의 수로 모듈러 연산을 하였습니다.

```
111  struct timer_list Jtime;
112  void tf(unsigned long data)
113  {
114      //to schedule a tasklet to run soon
115      tasklet_schedule(&name);
116      Jtime.expires = get_jiffies_64()+period;
117      add_timer(&Jtime);
118  }
```

다음은 타이머 함수에 들어가는 함수로, 타이머 발생시 실행되는 함수를 정의합니다. 타이머 함수 안에서는 tasklet 이 스케줄되며, 커멘드창에서 입력한 period 의 주기를 기준으로 타이머가 expire 되면, 다시 타이머에 타이머 리스트를 넣습니다.

```
119  //bitwise operation to get bit in corresponding bit
120  char* bit_checker(int bit, unsigned long padr) {
121      char* result = "";
122  switch(bit) {
123          case 7:
124              if(padr & 128) {
125                  result = "4MB";
126              }
127              else {
128                  result = "4KB";
129              }
130          break;
131          case 5:
132              if(padr & 32) {
133                  result = "1";
134              }
135              else {
136                  result = "0";
137              }
138          break;
139          case 4:
140              if(padr & 16) {
141                  result = "true";
142              }
143              else {
144                  result = "false";
145              }
146          break;
147          case 3:
148              if(padr & 8) {
149                  result = "write-through";
150              }
151              else {
152                  result = "write-back";
153              }
154          break;
155          case 2:
156              if(padr & 4) {
157                  result = "user";
158              }
159              else {
160                  result = "supervisor";
```

다음은 pgd 에 관련된 bit 를 확인하는 함수로써 해당 비트에 위치에 해당하는 bit 를 bit 연산자를 통해 값을 확인하고 결과값을 char*으로 리턴하는 함수입니다. 비트 연산자 & 로 해당 비트를 확인할 수 있습니다.

```
186  //bitwise operation to get bit in corresponding bit (pte)
187  char* bit_checker2(int bit, unsigned long padr) {
188      char* result = "";
189  switch(bit) {
190          case 6:
191              if(padr & 64) {
192                  result = "1";
193              }
194              else {
195                  result = "0";
196              }
197          break;
198          case 5:
199              if(padr & 32) {
200                  result = "1";
201              }
202              else {
203                  result = "0";
204              }
205          break;
206          case 4:
207              if(padr & 16) {
208                  result = "true";
209              }
210              else {
211                  result = "false";
212              }
213          break;
214          case 3:
215              if(padr & 8) {
216                  result = "write-through";
217              }
218              else {
219                  result = "write-back";
220              }
```

다음은 pte 에 관련된 bit 를 확인하는 함수로써 해당 비트에 위치에 해당하는 bit 를 bit 연산자를 통해 값을 확인하고 결과값을 char*으로 리턴하는 함수입니다. 비트 연산자 & 로 해당 비트를 확인할 수 있습니다.

```
256  static int my_seq_show(struct seq_file *s, void *v)
257  {
258
259      print_bar(s);
260      seq_printf(s, "Student ID: %s    Name: %s\n", STUDENT_ID, STUDENT_NAME);
261      seq_printf(s, "Virtual Memory Address Information\n");
262      seq_printf(s, "Process (%15s:%lu)\n", picked->comm, picked->pid);
263
264      seq_printf(s, "Last update time %llu ms\n", get_jiffies_64());
265      print_bar(s);
```

다음은 sequence 의 step 을 실행하는 함수로 과제 스펙에 원하는 결과값을 print 하는 함수입니다.

해당 스펙에 맞게 학번, 이름을 먼저 print 하고, 해당 process, pid 그리고 last update time 을 print 합니다.

```
267      // print info about each area
268
269      seq_printf(s, "0x%08lx - 0x%08lx : Code Area, %lu page(s)\n", picked->mm->start_code, picked->mm->end_code, (picked->mm->end_code - picked->mm->start_code)/PAGE_SIZE);
270      seq_printf(s, "0x%08lx - 0x%08lx : Data Area, %lu page(s)\n", picked->mm->start_data, picked->mm->end_data, (picked->mm->end_data - picked->mm->start_data)/PAGE_SIZE);
271      seq_printf(s, "0x%08lx - 0x%08lx : BSS Area, %lu page(s)\n", picked->mm->end_data, picked->mm->start_brk, (picked->mm->start_brk - picked->mm->end_data)/PAGE_SIZE);
272      seq_printf(s, "0x%08lx - 0x%08lx : Heap Area, %lu page(s)\n", picked->mm->start_brk, picked->mm->brk, (picked->mm->brk - picked->mm->start_brk)/PAGE_SIZE);
273
```

다음의 code area, data area, bss area, heap area 는 mm_types.h 의 mm_struct 에 정의 되어있는 값들을 활용하여 구할 수 있었습니다. 또한, 각 area 의 시작과 끝의 차를 PAGE_SIZE 로 나누어 해당 area 의 실질적 page 의 수도 구할 수 있었습니다.

```
273      //where are you guys..
274      //seq_printf(s, "0x%08lx - 0x%08lx : Shared Libraries Area, %lu page(s)\n", 0x7fc1f8225000, 0x7fc1fbadb000, picked->mm->shared_vm);
275      seq_printf(s, "0x%08lx - 0x%08lx : Shared Libraries Area, %lu page(s)\n", picked->mm->mmap->vm_start, picked->mm->mmap->vm_end, (picked->mm->mmap->vm_end - picked->mm->mmap
276      seq_printf(s, "0x%08lx - 0x%08lx : Stack Area, %lu page(s)\n", picked->mm->start_stack, end_of_stack(picked), (*end_of_stack(picked)-picked->mm->start_stack)/PAGE_SIZE);
277
```

Shared memory 의 경우 mmap 에 값이 vm_start 그리고 vm_end 의 값이 있어 다음의 값을 구할 수 있었습니다. Stack 의 경우 mm 에 start_stack 을 사용하여 시작주소를 구하였고, stack 의 끝 값은 구하는 방법을 찾지 못하여 stack pointer 가 가르키는 주소 값을 사용하였습니다.

```
278     // 1 level paging (PGD Info)
279     print_bar(s);
280     seq_printf(s, "1 Level Paging: Page Directory Entry Information \n");
281     print_bar(s);
282     //declare pgd table base address
283     pgd_t *j_pgd = picked->mm->pgd;
284     unsigned long linearAddr = picked->mm->start_code;
285     seq_printf(s, "PGD     Base Address            : 0x%08lx\n", j_pgd);
286     //pgd_offset macro to calculate
287     pgd_t *pgdAddr = pgd_offset(picked->mm, linearAddr);
288     seq_printf(s, "code    PGD Address             : 0x%08lx\n", pgdAddr);
289     unsigned long pgdV = pgdAddr->pgd;
290     seq_printf(s, "        PGD Value               : 0x%08lx\n", pgdV);
291     seq_printf(s, "        +PFN Address            : 0x%08lx\n", pgdV/PAGE_SIZE);
```

다음 1 level paging PGD info 를 구한 부분입니다. Base address 는 mm struct 의 pgd 를 사용하여 구하였습니다. Pgd 의 실질적 주소는 pgd_offset 매크로를 사용하여 구하였고, 그에 해당하는 pgd 주소는 pgd 가 포인트하는 주소값을 사용하여 구하였습니다. 또한 실질적 PFN 의 값의 경우 pgd 값을 해당 주소의 상태를 나타내는 부분을 PAGE_SIZE 로 나누어 실질적 주소값만 구할 수 있었습니다.

```
293     seq_printf(s, "        +Page Size              : %s\n", bit_checker(7, (unsigned long)pgdAddr));
294     seq_printf(s, "        +Accessed Bit           : %s\n", bit_checker(5, (unsigned long)pgdAddr));
295     seq_printf(s, "        +Cache Disable Bit      : %s\n", bit_checker(4, (unsigned long)pgdAddr));
296     seq_printf(s, "        +Page Write-Through     : %s\n", bit_checker(3, (unsigned long)pgdAddr));
297     seq_printf(s, "        +User/Supervisor Bit    : %s\n", bit_checker(2, (unsigned long)pgdAddr));
298     seq_printf(s, "        +Read/Write Bit         : %s\n", bit_checker(1, (unsigned long)pgdAddr));
299     seq_printf(s, "        +Page Present Bit       : %s\n", bit_checker(0, (unsigned long)pgdAddr));
300
```

다음은 해당 pgd 의 bit 를 확인하는 함수로 위에서 설명했던 함수를 사용하여 비트연산자를 통해 각 해당 비트를 구할 수 있었습니다.

```
301     // 2 level paging (PUD Info)
302     print_bar(s);
303     seq_printf(s, "2 Level Paging: Page Upper Directory Entry Information \n");
304     print_bar(s);
305     //declare pud table base address
306     pud_t *j_pud = pud_offset(pgdAddr, linearAddr);
307     seq_printf(s, "code     PUD Address             : 0x%08lx\n", j_pud);
308     unsigned long pudV = j_pud->pud;
309     seq_printf(s, "         PUD Value               : 0x%08lx\n", pudV);
310     seq_printf(s, "         +PFN Address            : 0x%08lx\n", pudV/PAGE_SIZE);
311
312     // 3 level paging (PMD Info)
313     print_bar(s);
314     seq_printf(s, "3 Level Paging: Page Middle Directory Entry Information \n");
315     print_bar(s);
316     //declare pmd table base address
317     pmd_t *j_pmd = pmd_offset(j_pud, linearAddr);
318     seq_printf(s, "code     PMD Address             : 0x%08lx\n", j_pmd);
319     unsigned long pmdV = j_pmd->pmd;
320     seq_printf(s, "         PMD Value               : 0x%08lx\n", pmdV);
321     seq_printf(s, "         +PFN Address            : 0x%08lx\n", pmdV/PAGE_SIZE);
322
323     // 4 level paging (PTE Info)
324     print_bar(s);
325     seq_printf(s, "4 Level Paging: Page Table Entry Information \n");
326     print_bar(s);
327     //delcare pte table base address
328     pte_t *j_pte = pte_offset_kernel(j_pmd, linearAddr);
329
330     seq_printf(s, "code     PTE Address             : 0x%08lx\n", j_pte);
331     unsigned long pteV = j_pte->pte;
332     seq_printf(s, "         PTE Value               : 0x%08lx\n", pteV);
333     seq_printf(s, "         +Page Base Address      : 0x%08lx\n", pteV/PAGE_SIZE);
334
335     seq_printf(s, "         +Dirty Bit              : %s\n", bit_checker2(6,(unsigned long)j_pte));
336     seq_printf(s, "         +Accessed Bit           : %s\n", bit_checker2(5,(unsigned long)j_pte));
337     seq_printf(s, "         +Cache Disable Bit      : %s\n", bit_checker2(4,(unsigned long)j_pte));
338     seq_printf(s, "         +Page Write-Through     : %s\n", bit_checker2(3,(unsigned long)j_pte));
339     seq_printf(s, "         +User/Supervisor        : %s\n", bit_checker2(2,(unsigned long)j_pte));
340     seq_printf(s, "         +Read/Write Bit         : %s\n", bit_checker2(1,(unsigned long)j_pte));
341     seq_printf(s, "         +Page Present Bit       : %s\n", bit_checker2(0,(unsigned long)j_pte));
```

다음은 pgd 를 구했던 방식으로 pud,pmd pte 에 해당하는 값을 구한 것입니다. Pte 의 경우 bit_checker2 를 통해 pge 에서 처럼 해당 비트의 의미를 알 수 있습니다.

```
343     unsigned long phys = pteV & PAGE_MASK;
344     unsigned long phys_off = linearAddr & ~PAGE_MASK;
345     print_bar(s);
346     seq_printf(s, "Start of Physical Address       : 0x%08lx\n", phys | phys_off);
347     print_bar(s);
348     seq_printf(s, "Start of Virtual Address        : 0x%08lx\n", phys_to_virt(phys | phys_off));
349     print_bar(s);
350
```

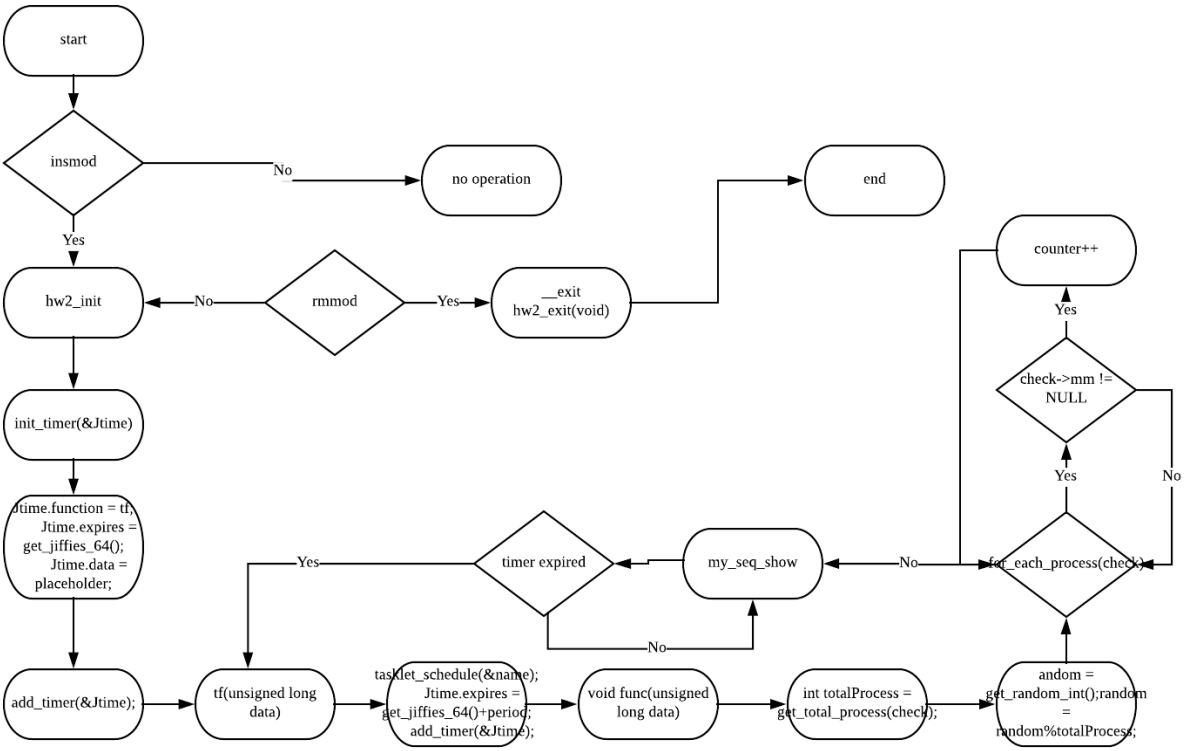
Physical Address 는 linear address 와 ~PAGE_MASK 의 &로 구한 비트와 pte 값 과 PAGE_MASK 의 주소를 |로 더하여 나타냈습니다.

Virtual Address 는 phys_to_virt 함수를 사용하여 나타냈습니다.

```
387  static int __init hw2_init(void) {
388
389
390      init_timer(&Jtime);
391      Jtime.function = tf;
392      Jtime.expires = get_jiffies_64();
393      Jtime.data = placeholder;
394      add_timer(&Jtime);
395
396      struct proc_dir_entry *entry;
397      entry = proc_create(PROC_NAME, 0, NULL, &my_file_ops);
398
399      printk(KERN_INFO "TESTING\n");
400      return 0;
401  }
402
403
404
405  static void __exit hw2_exit(void) {
406      remove_proc_entry(PROC_NAME, NULL);
407      del_timer(&Jtime);
408      printk(KERN_INFO "TESTING ENDS\n");
409  }
410
```

다음은 실질적으로 insmod 를 하여 timer 가 시작되는 부분이고, rmmod 를 할 때 제거되야 하는 것을 나타내는 부분입니다.

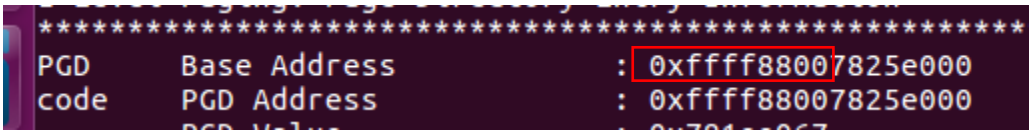
작성한 프로그램의 동작 과정의 순서도

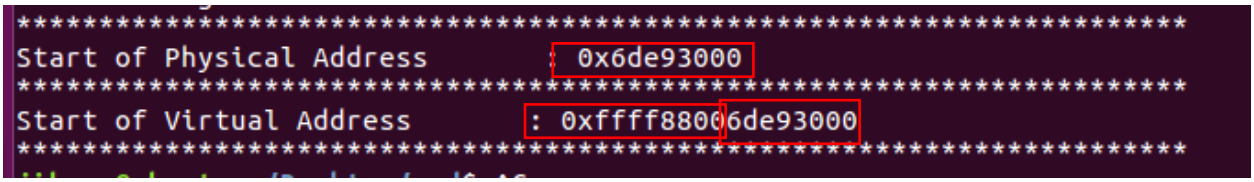


Physical memory 를 virtual memory 로 바꾸는 함수 분석

```
87  static inline void * phys_to_virt(unsigned long address)
88  {
89          return (void *)(IDENT_ADDR + (address & ((1ul << 41) - 1)));
90  }
91  #endif
```

현재 physical address 의 해당 주소값과 baseaddress 의 앞 부분을 비트 연산자로 더하여 다음의 결과값을 얻습니다.

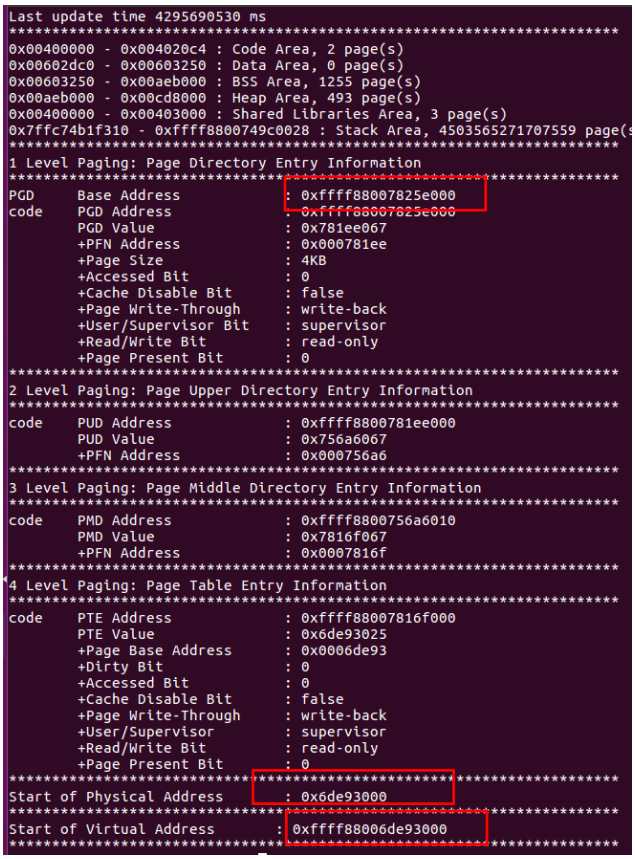




Physical address 와 base address 의 비트 연산을 통한 결과가 나옵니다.

```
21  /*
22   * Virtual -> physical identity mapping starts at this offset
23   */
24  #ifdef USE_48_BIT_KSEG
25  #define IDENT_ADDR      0xffff800000000000UL
26  #else
27  #define IDENT_ADDR      0xfffffc0000000000UL
28  #endif
```

처음 시작된 virtual memory 와 physical memory 로 부터 역산한 virtual memory 의 값의 차이 (결과물의 이유 서술)



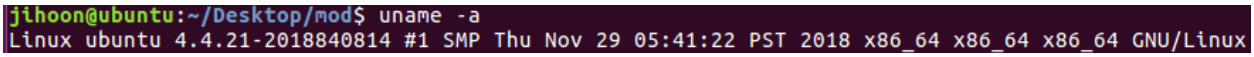
```
Last update time 4295690530 ms
*********************************************************************
0x00400000 - 0x004020c4 : Code Area, 2 page(s)
0x00602dc0 - 0x00603250 : Data Area, 0 page(s)
0x00603250 - 0x00aeb000 : BSS Area, 1255 page(s)
0x00aeb000 - 0x00cd8000 : Heap Area, 493 page(s)
0x00400000 - 0x00403000 : Shared Libraries Area, 3 page(s)
0x7ffc74b1f310 - 0xffff8800749c0028 : Stack Area, 4503565271707559 page(s
*********************************************************************
1 Level Paging: Page Directory Entry Information
*********************************************************************
PGD     Base Address            : 0xffff88007825e000
code    PGD Address             : 0xffff88007825e000
        PGD Value               : 0x781ee067
        +PFN Address            : 0x000781ee
        +Page Size              : 4KB
        +Accessed Bit           : 0
        +Cache Disable Bit      : false
        +Page Write-Through     : write-back
        +User/Supervisor Bit    : supervisor
        +Read/Write Bit         : read-only
        +Page Present Bit       : 0
*********************************************************************
2 Level Paging: Page Upper Directory Entry Information
*********************************************************************
code    PUD Address             : 0xffff8800781ee000
        PUD Value               : 0x756a6067
        +PFN Address            : 0x000756a6
*********************************************************************
3 Level Paging: Page Middle Directory Entry Information
*********************************************************************
code    PMD Address             : 0xffff8800756a6010
        PMD Value               : 0x7816f067
        +PFN Address            : 0x0007816f
*********************************************************************
4 Level Paging: Page Table Entry Information
*********************************************************************
code    PTE Address             : 0xffff88007816f000
        PTE Value               : 0x6de93025
        +Page Base Address      : 0x0006de93
        +Dirty Bit              : 0
        +Accessed Bit           : 0
        +Cache Disable Bit      : false
        +Page Write-Through     : write-back
        +User/Supervisor        : supervisor
        +Read/Write Bit         : read-only
        +Page Present Bit       : 0
*********************************************************************
Start of Physical Address       : 0x6de93000
*********************************************************************
Start of Virtual Address        : 0xffff88006de93000
*********************************************************************
```

```
87  static inline void * phys_to_virt(unsigned long address)
88  {
89          return (void *)(IDENT_ADDR + (address & ((1ul << 41) - 1)));
90  }
91  #endif
```

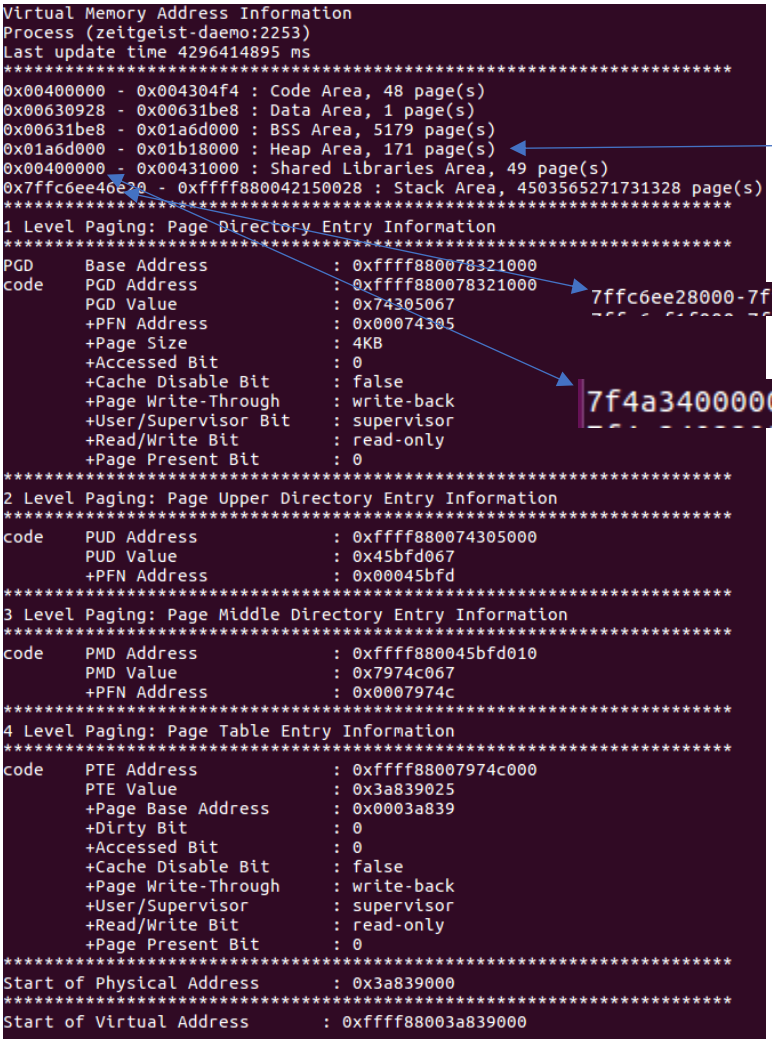
다음의 함수를 통해 virtual memory 의 값을 계산하지만, 예상과는 달리 실질적으로 대입한 virtual memory 의 값과는 다른 값이 나옵니다. 현재 physical address 의 해당 주소값과 base address 의 앞 부분을 비트 연산자로 더하여 다음의 결과값을 얻습니다. 하지만 IDENT_ADDR 의 값의 경우 디폴트로 다음의 값이 정해져 있어 다음의 함수를 사용하여 계산할 때 오차가 발생합니다.

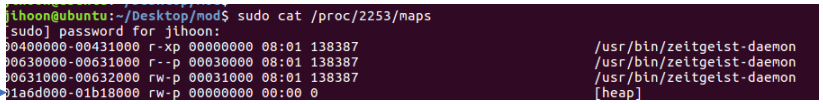
```
21  /*
22   * Virtual -> physical identity mapping starts at this offset
23   */
24  #ifdef USE_48_BIT_KSEG
25  #define IDENT_ADDR      0xffff800000000000UL
26  #else
27  #define IDENT_ADDR      0xffffffc0000000000UL
28  #endif
```

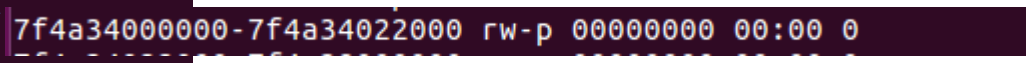
Uname -a 실행 결과 화면과 개발 환경을 명시한다.

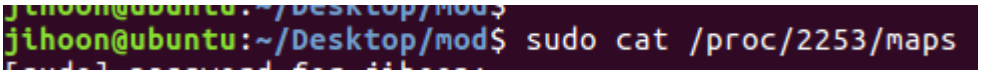


결과 화면과 결과에 대한 토의 내용









다음의 커멘드를 통해 해당 process 를 분석 하였습니다. 결과물의 경우 shared library 를 제외한 나머지 결과물은 상당 수 일치하는 결과를 얻을 수 있었습니다. Shared library 의 경우 heap 과 stack 사이의 값을 계산하는 아이디어를 찾지 못해 아쉬움이 남습니다.

참조

https://www.linuxquestions.org/questions/programming-9/random-numbers-kernel-642087/

https://stackoverflow.com/questions/16975393/current-mm-gives-null-in-linux-kernel

https://www.oreilly.com/library/view/linux-device-drivers/0596000081/ch09s05.html

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/8_MainMemory.html

http://egloos.zum.com/bigs/v/6073753

http://pobimoon.tistory.com/entry/%EB%A6%AC%EB%88%85%EC%8A%A4%EC%97%90%EC%84%9C-%EC%82%AC%EC%9A%A9%EB%90%98%EB%8A%94-%EB%A9%94%EB%AA%A8%EB%A6%AC-%EB%AA%A8%EB%8D%B8%EC%9D%84-%EC%9D%B4%ED%95%B4

https://github.com/torvalds/linux/blob/master/include/linux/random.h

http://byeoksan.tistory.com/entry/Chapter-2-Memory-Addressing

http://jake.dothome.co.kr/buddy-alloc/