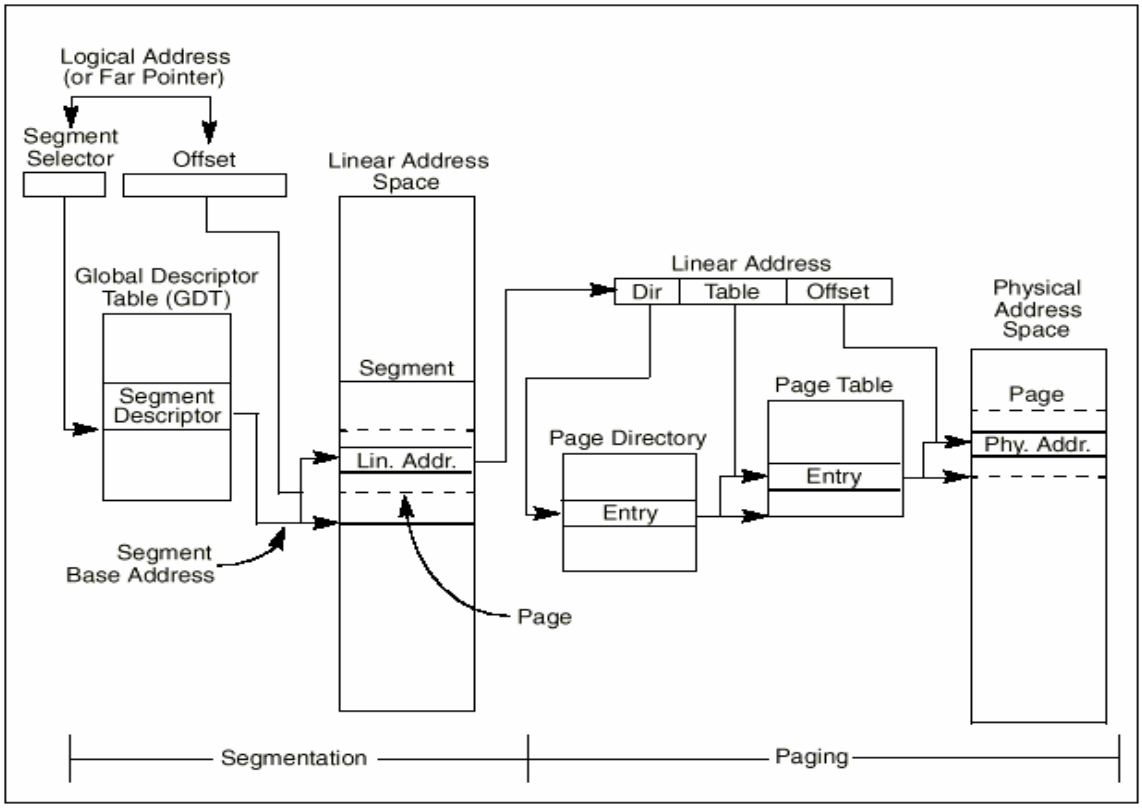
Linux Memory Management

Source : 이 형주님
kerdosa@hotmail.com
www.kesl.org

목 차

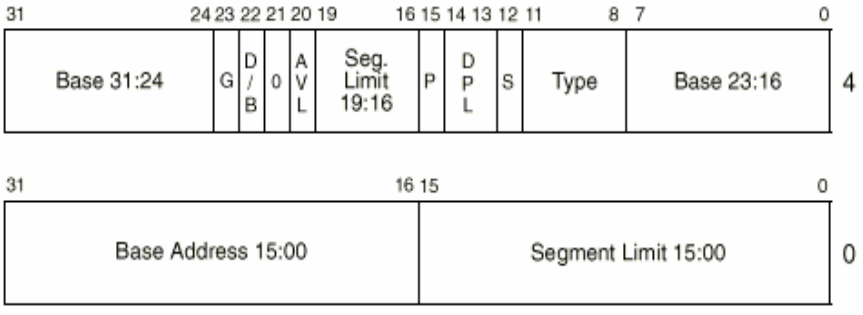
1-1. i386 : Segmentation & Paging



1-2. Segment Descriptor

-base, limit, type, dpl

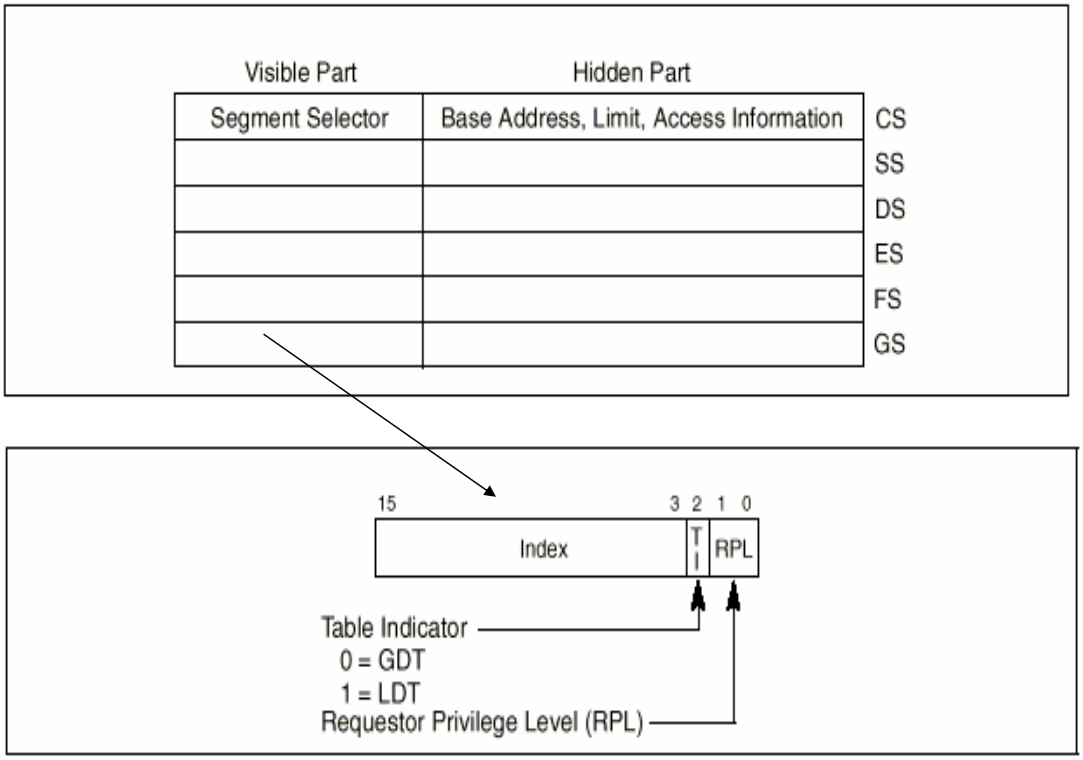
-ex : 0xc0c39a000000ffff /* 0x10 kernel 1GB code*/



AVL—Available for use by system software
BASE—Segment base address
D/B—Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL—Descriptor privilege level
G—Granularity
LIMIT—Segment Limit
P—Segment present
S—Descriptor type (0= system; 1 = application)
TYPE—Segment type

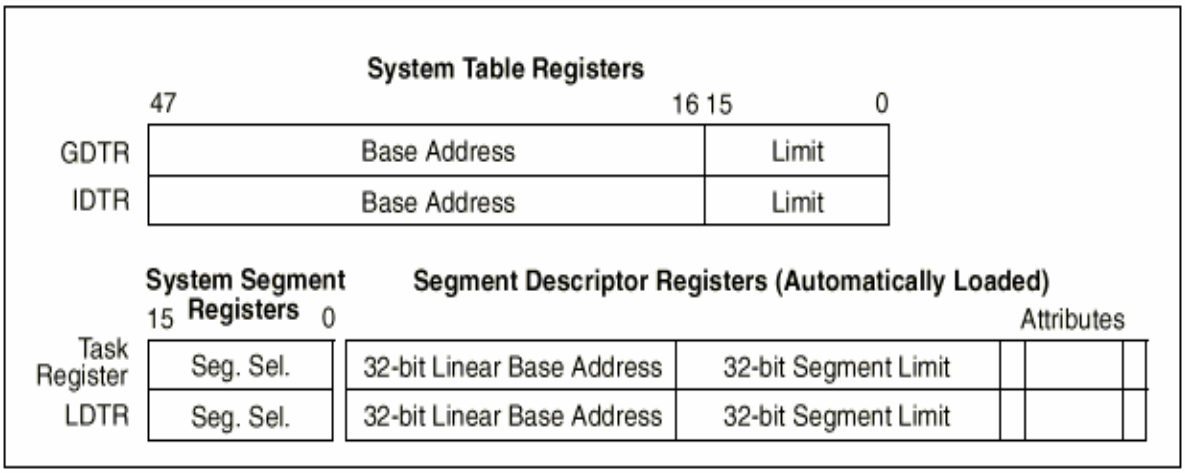
1-3. Segment Selector

cs = __KERNEL_CS /* 0x10 */



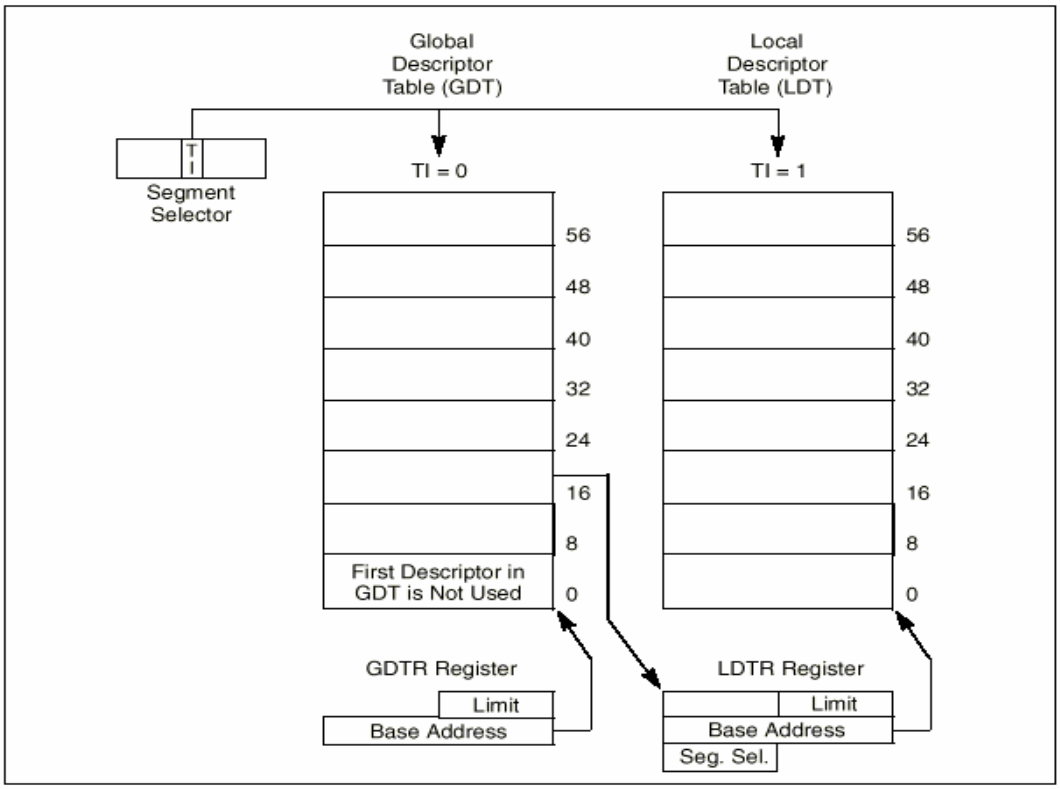
1-4. System Register

-명령어 : lgdt, lldt, ltr, lidt.
-gdtr, idtr는 booting시 head.S에서 한번만 load된다.
-ldtr, tr는 task switching시 gdt에 있는 해당 task's ldt, tss descriptor을 가리킨다.
-gdt, idt의 최대 entry는 8191 = 0xFFFF/8



1-5. GDT & LDT

-0x10 : gdt에서 3번째 descriptor



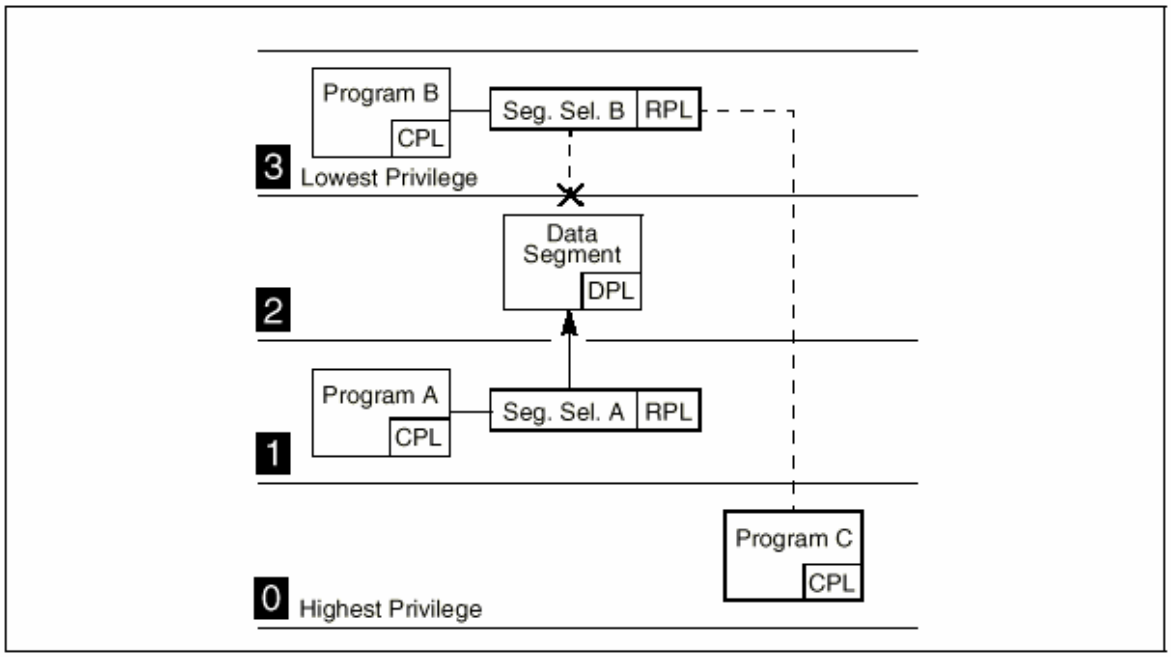
1-6. Protection Mechanism

-Linux, Win95 : 32bit 보호모드 O.S
-segment level, page level
-When violation, GP fault 발생

- **Limit checks**
- **Type checks**
  - CS register : only code segment
  - SS register : must writeable
- **Privilege level checks (Ring 0/1/2/3)**
  - CPL
  - RPL
  - DPL
- **Privileged instructions :** lgdt, lidt
- **Page level protection**
  - Restriction of addressable domain : supervisor/user
  - Page type : Read-only or Read/Write)

1-7. Privilege Level Check

If MAX(CPL,RPL) =< DPL, access allowed



<Privilege check for accessing data segment>

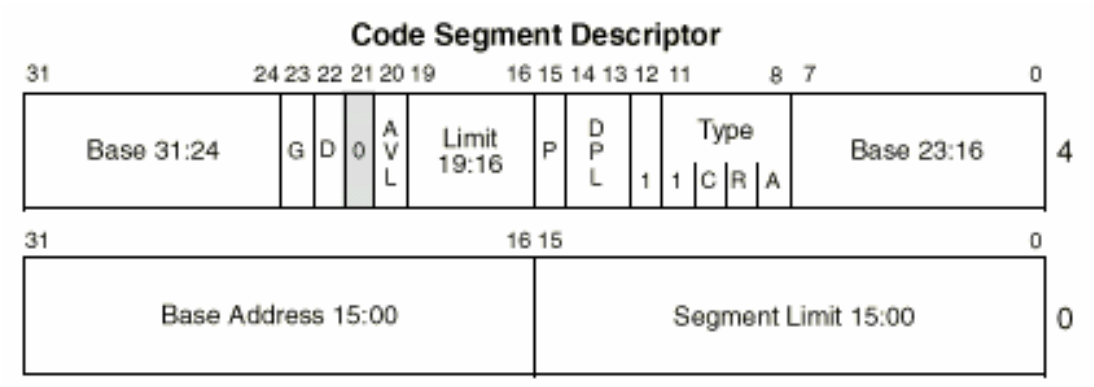
2-1. Linux Implementation : *flat memory model*

<list 1. arch/i386/kernel/head.S -- kernel 2.1.8 이전>

```
        lgdt gdt_descr              /* load gdtr register */
gdt_descr:
        .word (8+2*NR_TASKS)*8-1
SYMBOL_NAME(gdt):
        .long SYMBOL_NAME(gdt_table)

ENTRY(gdt_table)
.quad 0x0000000000000000        /* NULL descriptor */
.quad 0x0000000000000000        /* not used */
.quad 0xc0c39a000000ffff        /* 0x10 kernel 1GB code*/
.quad 0xc0c392000000ffff        /* 0x18 kernel 1GB data */
.quad 0x00cbfa000000ffff        /* 0x23 user   3GB code */
.quad 0x00cbf2000000ffff        /* 0x2b user   3GB data  */
.quad 0x0000000000000000        /* not used */
.quad 0x0000000000000000        /* not used */
.fill 2*NR_TASKS,8,0            /* space for LDT's and TSS's */
```

2-2. Example of Descriptor



/* 0x10 kernel 1GB code*/

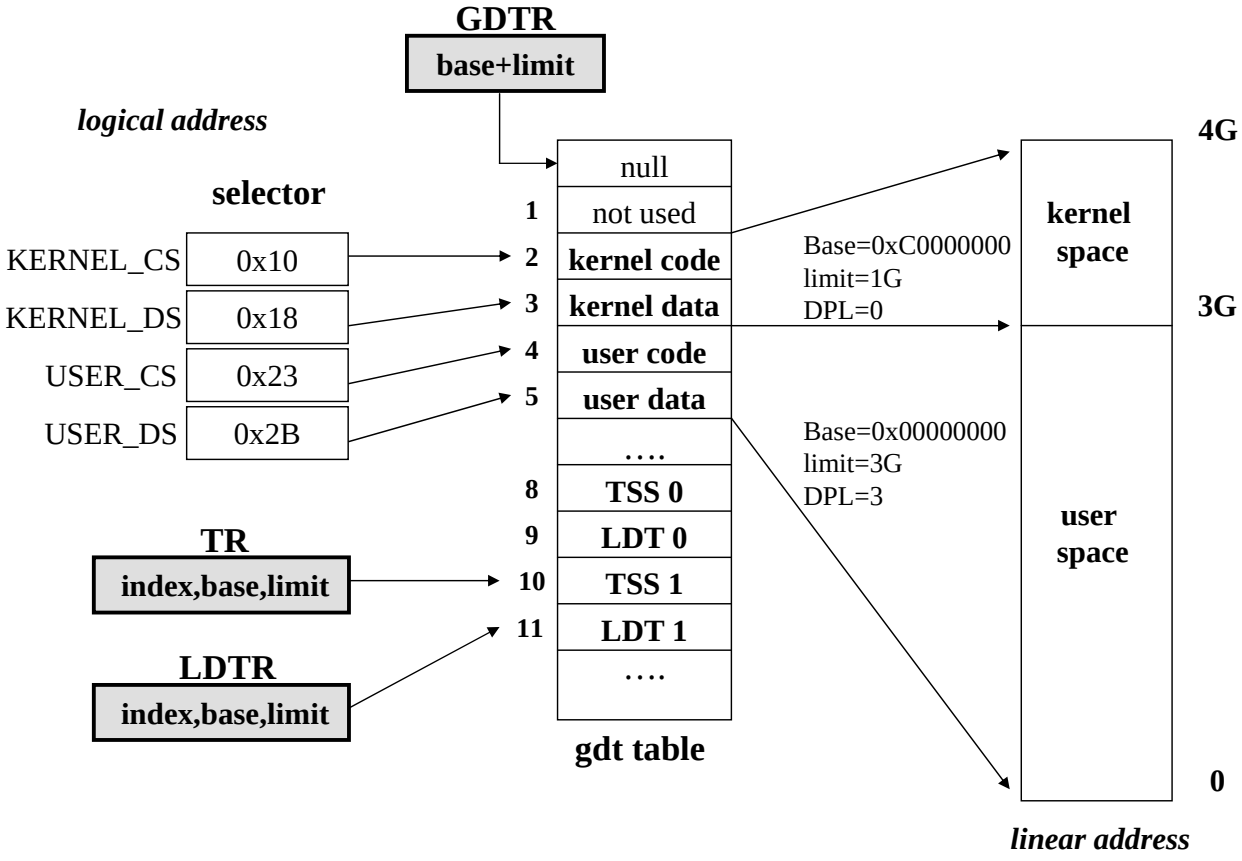
0xC0C3,9A00,0000,FFFF

C0C3,9A00,
0000,FFFF

00cb,fa00,
0000,ffff

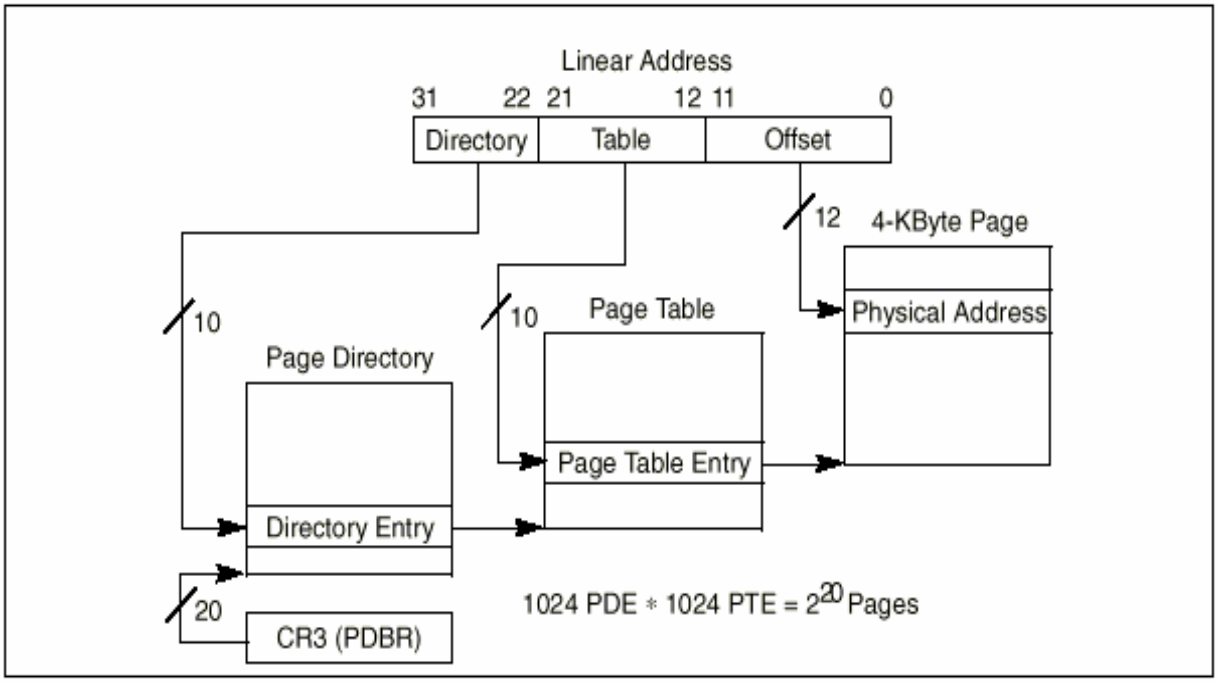
-Base : C000,0000
-Limit : 3FFFF*4K → 1GB
-Type : A → code, non-conforming , execute/read, not accessed
-S : 1 → app(code,data,stack)
-DPL(descriptor privilege level) : 00 → ring 0
-P : 1 → present
-G : 1 → 4k unit
-D/B : 1 → 32bit segment

2-3. Segmentation of Linux

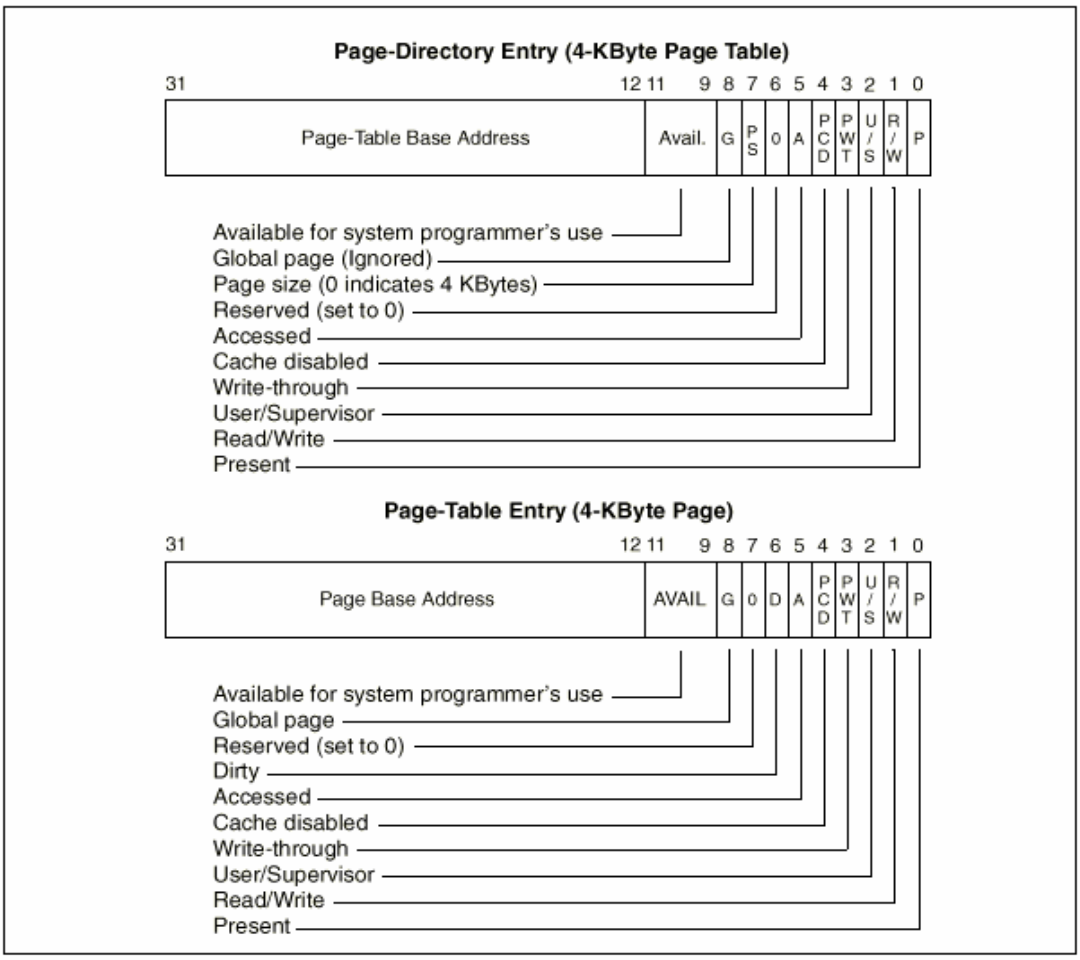


3-1. Address Translation by Paging

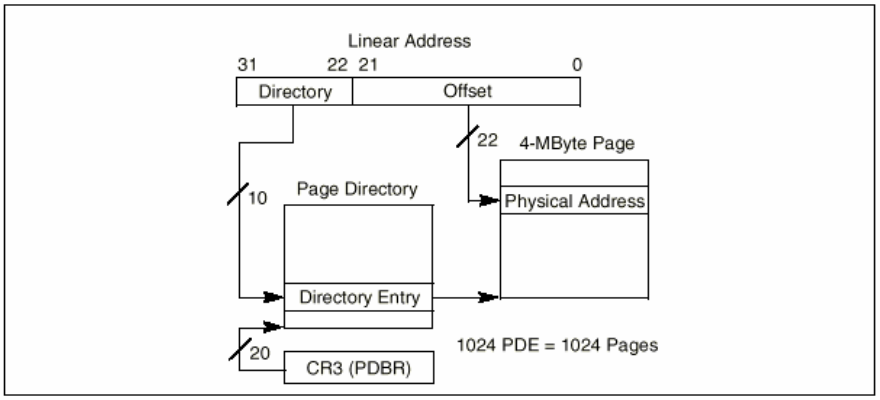
-4k paging



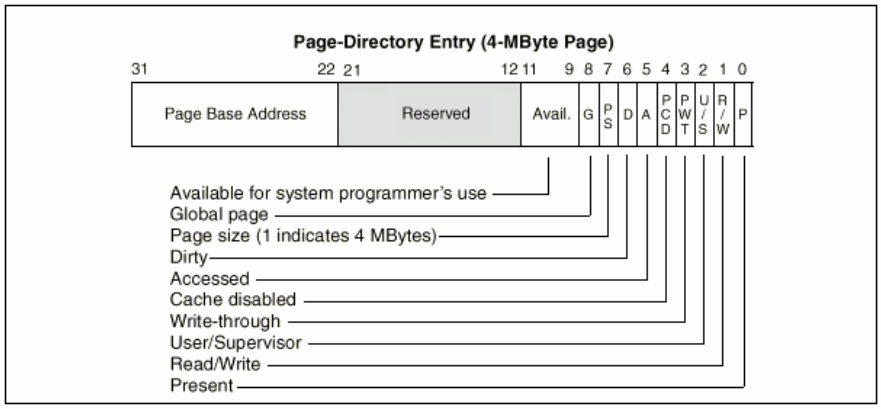
3-2. PGD & PTE



3-3. 4M Paging :

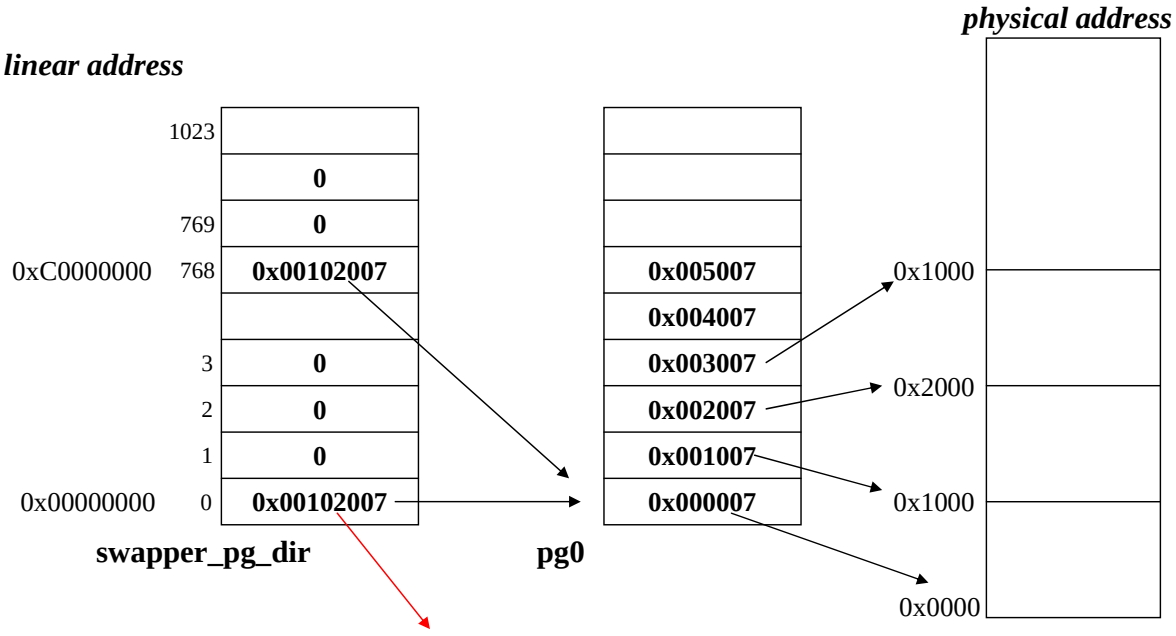


-Setting PSE[cr4]
& PS[pgd]



3-4. Linux Implementation

-head.S에서 초기 4M에 대한 page만 초기화한다.



-present,r/w,user,w/b,cached,not accessed,4k page,no global
-pte base : 0x00102000

3-5. Real value of swapper_pg_dir (64M)

0xc0101000 :	0x00000000	0x00000000	0x00000000	0x00000000
0xc0101010 :	0x00000000	0x00000000	0x00000000	0x00000000
		- - -		
0xc0101bd0 :	0x00000000	0x00000000	0x00000000	0x00000000
0xc0101be0 :	0x00000000	0x00000000	0x00000000	0x00000000
0xc0101bf0 :	0x00000000	0x00000000	0x00000000	0x00000000
0xc0101c00 :	**0x000001e3**	**0x004001e3**	**0x008001e3**	**0x00c001e3**
0xc0101c10 :	**0x010001e3**	**0x014001e3**	**0x018001e3**	**0x01c001e3**
0xc0101c20 :	**0x020001e3**	**0x024001e3**	**0x028001e3**	**0x02c001e3**
0xc0101c30 :	**0x030001e3**	**0x034001e3**	**0x038001e3**	**0x03c001e3**
0xc0101c40 :	***0x0383c063***	0x00000000	0x00000000	0x00000000
0xc0101c50 :	0x00000000	0x00000000	0x00000000	0x00000000
		- - -		
0xc0101fe0 :	0x00000000	0x00000000	0x00000000	0x00000000
0xc0101ff0 :	0x00000000	0x00000000	0x00000000	0x00000000

-present,r/w,supervisor,w/b,cached,accessed,4M paging, page base:0x0000,0000

3-6. Real value of pg0

0xc0102000 :	0x00000007	0x00001007	0x00002007	0x00003007
0xc0102010 :	0x00004007	0x00005007	0x00006007	0x00007007
0xc0102020 :	0x00008007	0x00009007	0x0000a007	0x0000b007
0xc0102030 :	0x0000c007	0x0000d007	0x0000e007	0x0000f007
0xc0102040 :	0x00010007	0x00011007	0x00012007	0x00013007
0xc0102050 :	0x00014007	0x00015007	0x00016007	0x00017007
0xc0102060 :	0x00018007	0x00019007	0x0001a007	0x0001b007
0xc0102070 :	0x0001c007	0x0001d007	0x0001e007	0x0001f007
0xc0102080 :	0x00020007	0x00021007	0x00022007	0x00023007
0xc0102090 :	0x00024007	0x00025007	0x00026007	0x00027007
0xc01020a0 :	0x00028007	0x00029007	0x0002a007	0x0002b007
-				
0xc0102fb0 :	0x003ec007	0x003ed007	0x003ee007	0x003ef007
0xc0102fc0 :	0x003f0007	0x003f1007	0x003f2007	0x003f3007
0xc0102fd0 :	0x003f4007	0x003f5007	0x003f6007	0x003f7007
0xc0102fe0 :	0x003f8007	0x003f9007	0x003fa007	0x003fb007
0xc0102ff0 :	0x003fc007	0x003fd007	0x003fe007	0x003ff007

4-1. Memory Initailization

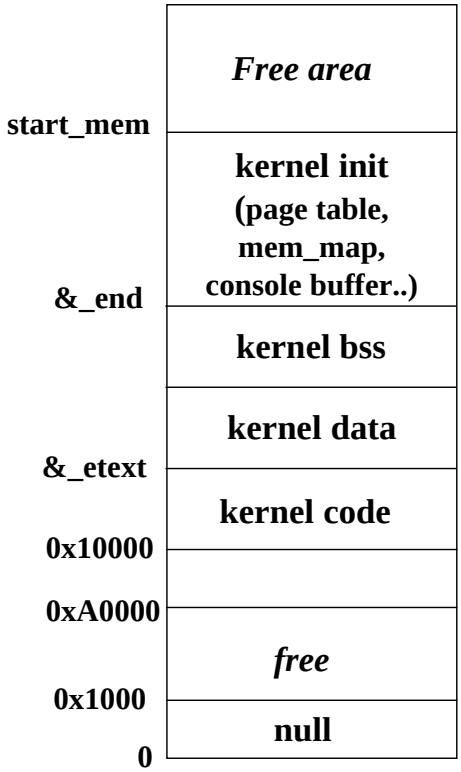
main.c: start_kernel()

↓

setup_arch() : memory_start, memory_end

paging_init() :kernel pgd, pte 초기화
free_area_init() :
-mem_map, bitmap memory 할당
-count=0, PG_Reserved 설정
-frea_area[I].map = &bitmap;

-mem_init() **:**
-start_mem~end_mem : clear PG_Reserved
-code,data,init,reserved,free page count & display
-free_page() : free_area[], bitmap 초기화



Booting message :
Memory: 14380k/16384k available (1064k kernel code, 412k reserved, 464k data, 64k init)

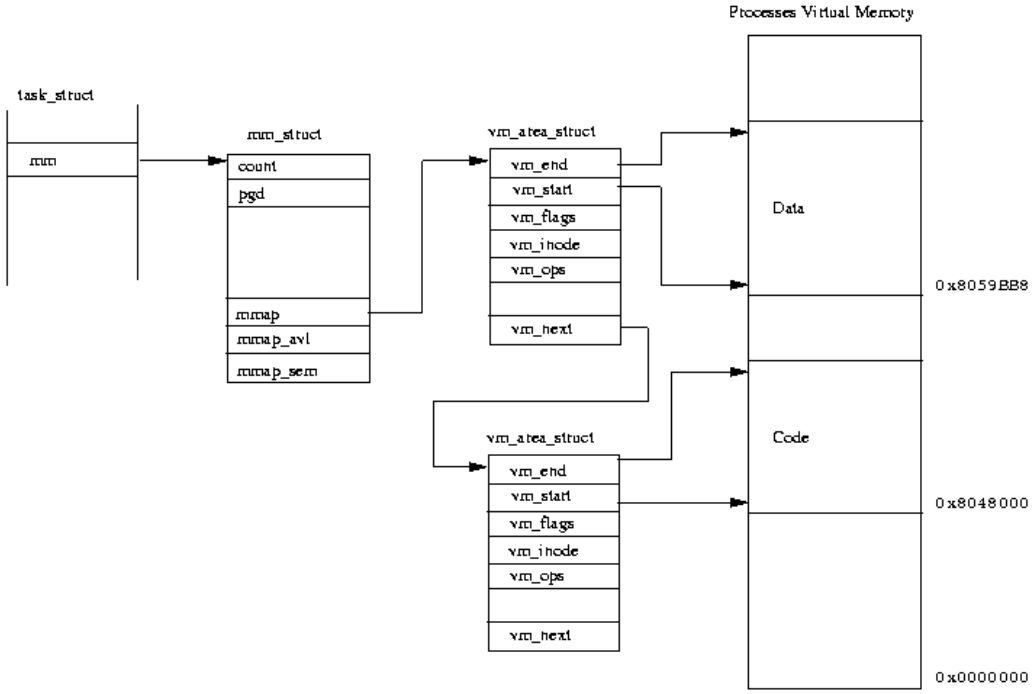
5-1. Process Virtual Address Space

-각 process는 fork()시 자기의 page directory을 가진다.
-각 process는 4G virtual address공간에서 실행된다.
-vm_area_struct 구조체로 연결되어 있다.

```
$cat /proc/1/maps

start-end     permission ofset maj:min inode
-------------------------------------------------------------
08048000-0804e000 r-xp 00000000 03:02 6224  #/sbin/init - code
0804e000-0804f000 rw-p 00005000 03:02 6224  #/sbin/init - data
0804f000-08053000 rwxp 00000000 00:00 0     # init bss
40000000-40012000 r-xp 00000000 03:02 4058  # /lib/ld-2.1.1.so
40012000-40013000 rw-p 00011000 03:02 4058  # /lib/ld-2.1.1.so
40013000-40014000 rwxp 00000000 00:00 0     # ld bss
bfffe000-c0000000 rwxp fffff000 00:00 0     # stack
```

5-2. Process Virtual Address Space

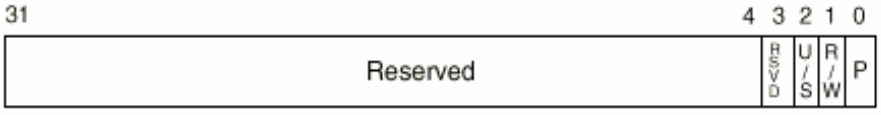


6-1. Page Falut

Interrupt & Exception의 종류

1. **S/W interrupt : system call**
2. **H/W interrupt by I/O**
3. **CPU exception : trap, fault, abort**

<error code of page fault>



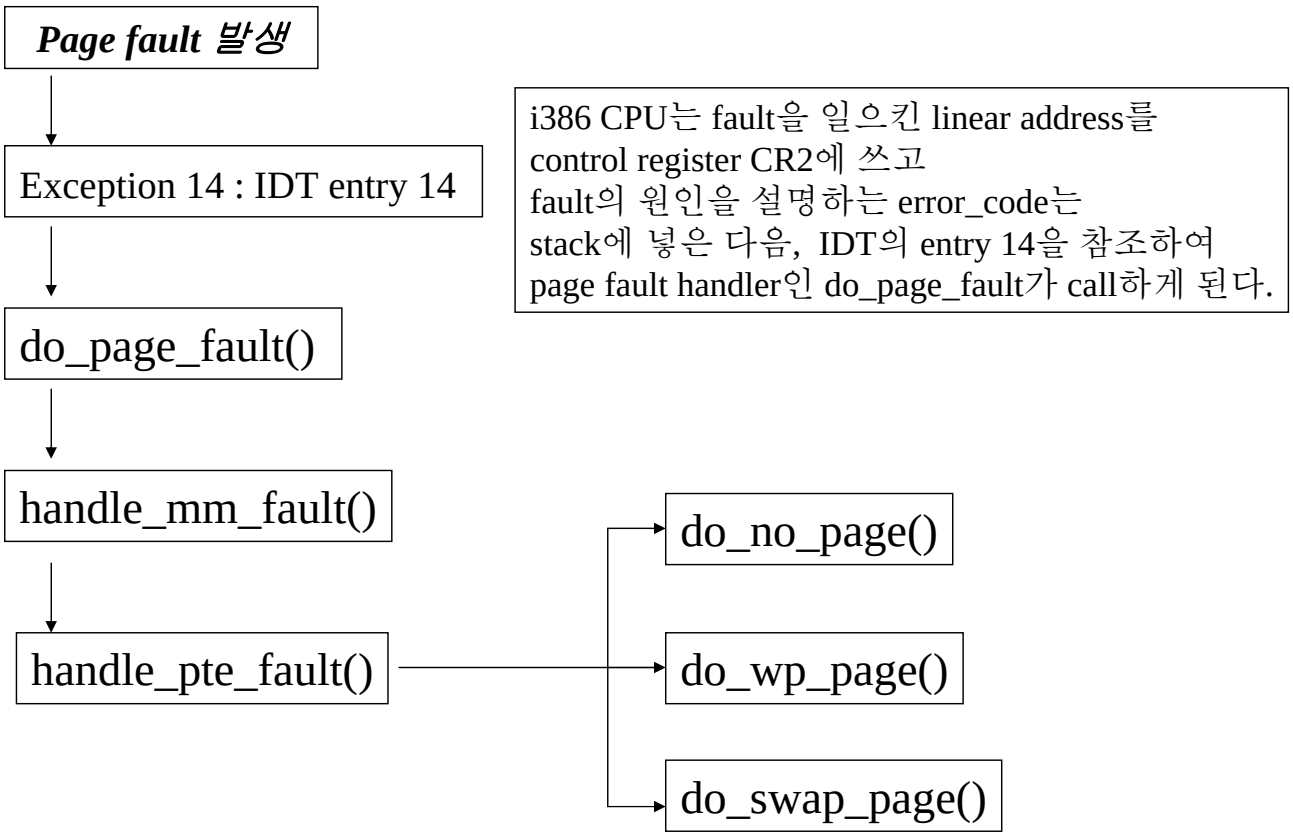
P 0 The fault was caused by a non-present page.
1 The fault was caused by a page-level protection violation.

W/R 0 The access causing the fault was a read.
1 The access causing the fault was a write.

U/S 0 The access causing the fault originated when the processor was executing in supervisor mode.
1 The access causing the fault originated when the processor was executing in user mode.

RSVD 0 The fault was caused by reserved bits set to 1 in a page directory
1 The fault was not caused reserved bit violation.

6-1. Page Fault Handler



7-1. User Space Access

-kernel 2.1.8 이전

-fs segment register 사용
-kernel mode 진입시
SAVE_ALL,
user mode로 복귀시
RESTORE_ALL을 실행

-get_user(), put_user()
-verify_area()

```
#define SAVE_ALL \
        cld; \
        push %gs; \
        push %fs; \
        push %es; \
        push %ds; \
        pushl %eax; \
        pushl %ebp; \
        pushl %edi; \
        pushl %esi; \
        pushl %edx; \
        pushl %ecx; \
        pushl %ebx; \
        movl $(KERNEL_DS),%edx; \
        mov %dx,%ds; \
        mov %dx,%es; \
        movl $(USER_DS),%edx; \
        mov %dx,%fs;
```

7-1. Reference

- **Linux Kernel Source**

- **LDP 문서**
- **Kernel Hacker's Guide**
- **The Linux Kernel**

- **Linux kernel Internal**
- **Linux Device Driver**
- **Advanced Programming in the Unix Environment**

- **Protected Mode Sofrware Archtecture**
- **Pentium ® Pro Family Developer's Manual Volume 3: Operating System Writer's Guide**