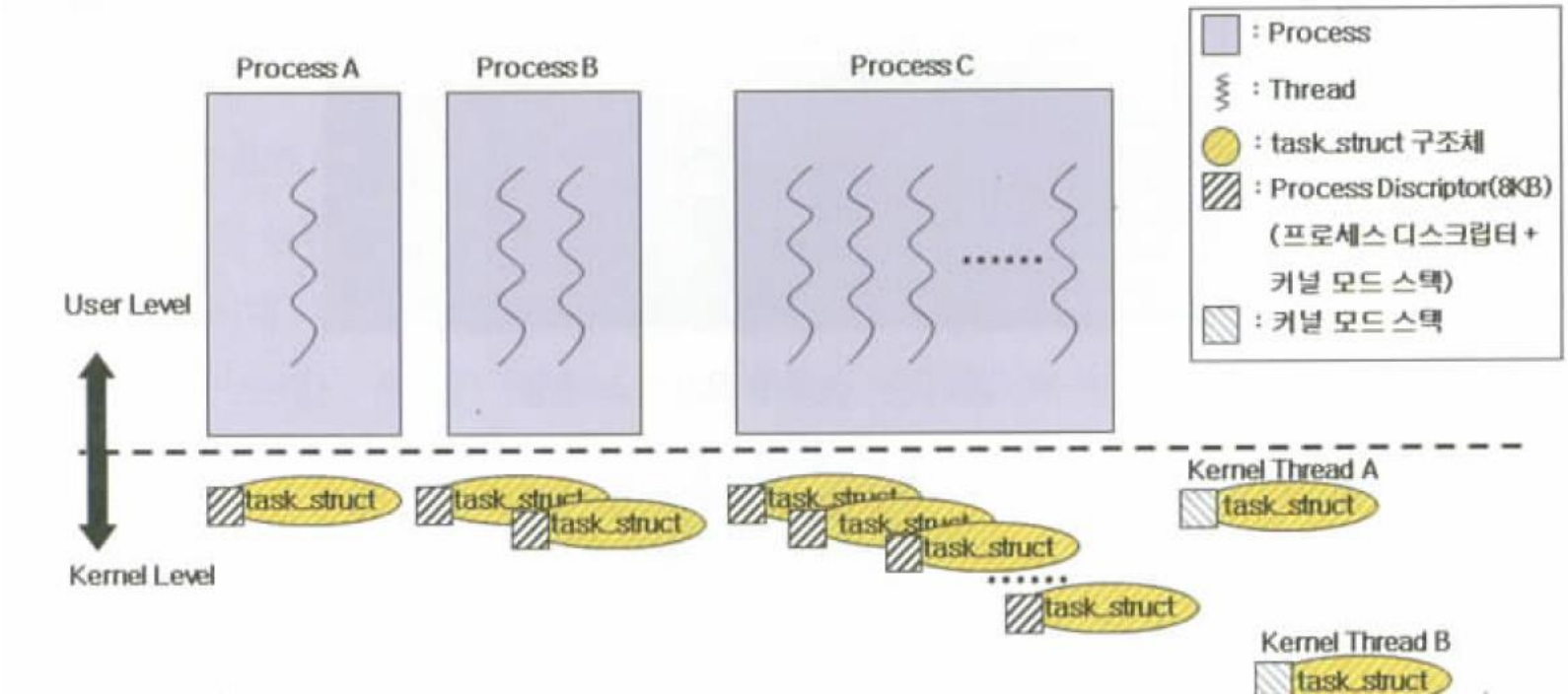
Operating System 실습
[Memory Management]

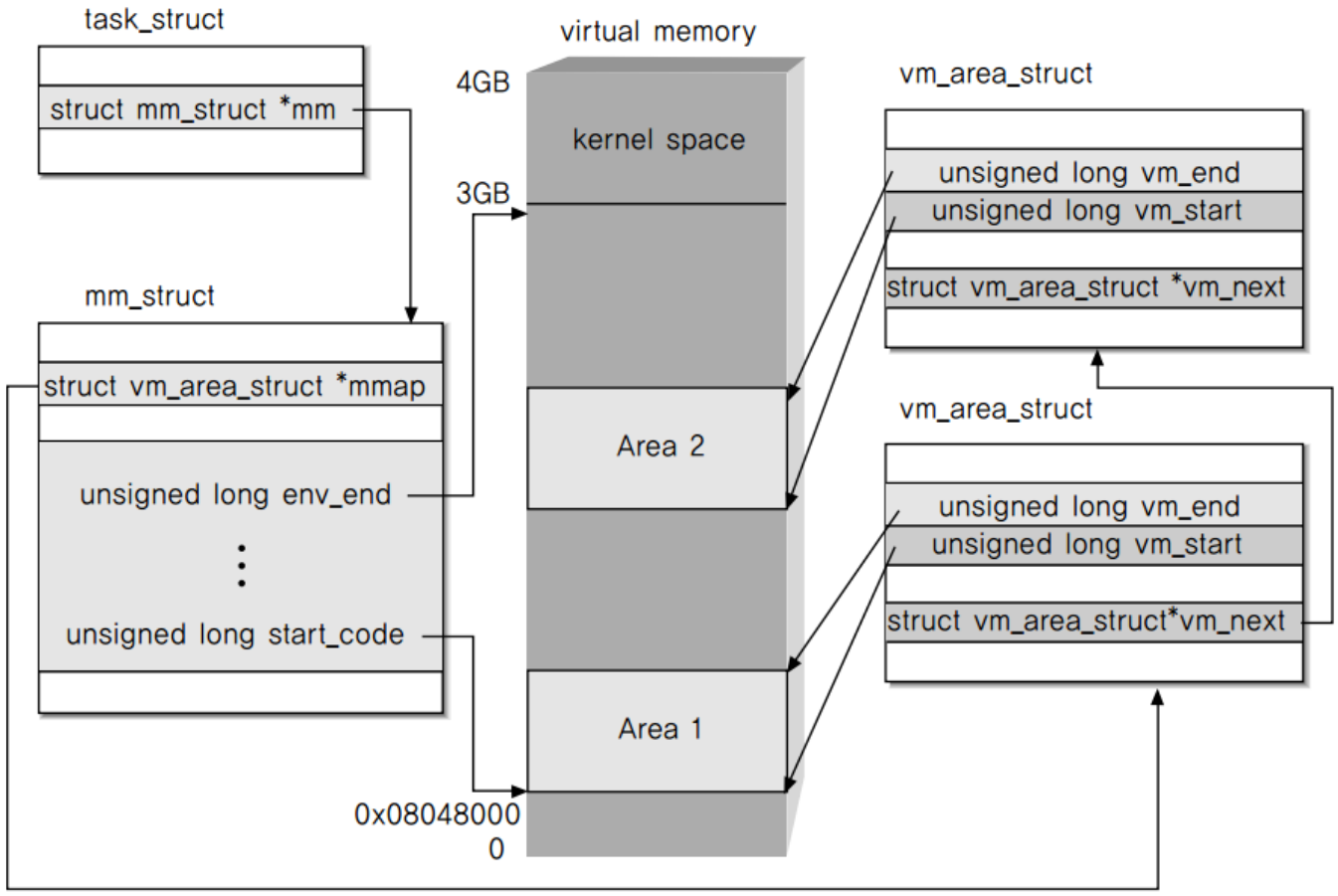
한양대학교
HANYANG UNIVERSITY

Data-Centric Systems Lab.

Memory Management/Virtual Memory



Memory Management/Virtual Memory

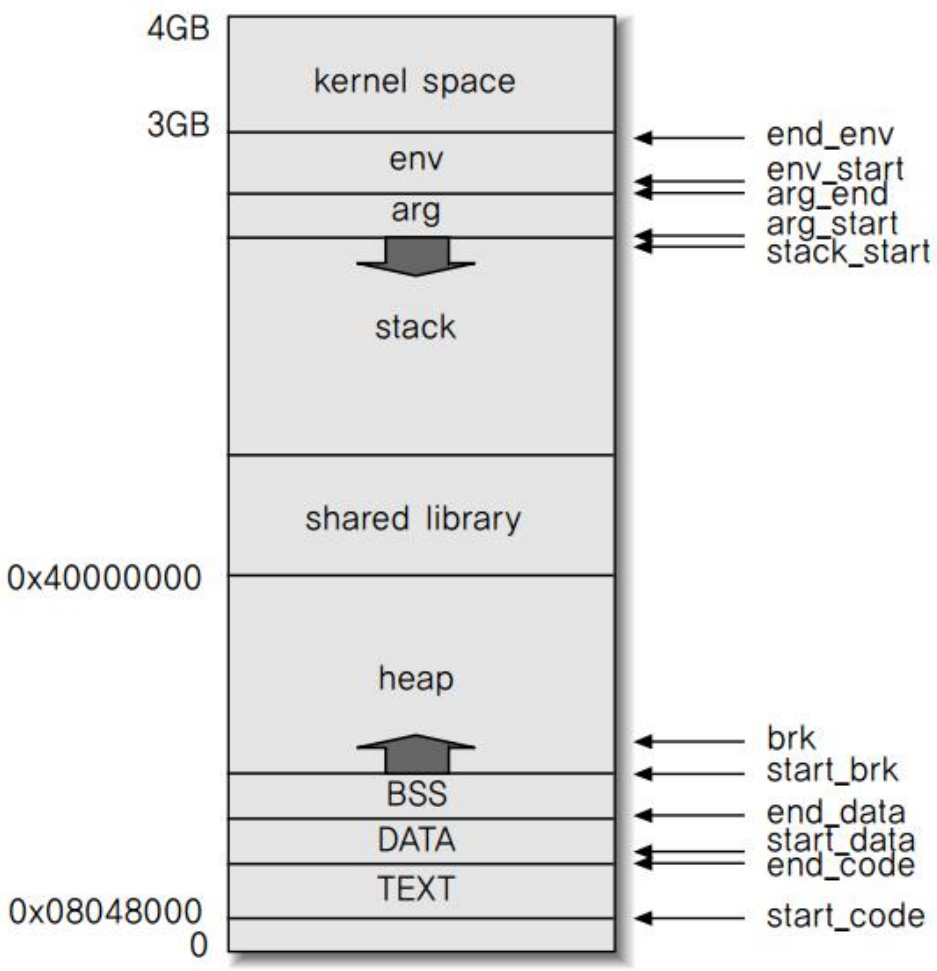


<가상 메모리 관리(32bit)>

Memory Management/Virtual Memory

```
struct mm_struct {
        struct vm_area_struct * mmap;
        rb_root_t mm_rb;
        struct vm_area_struct * mmap_cache;
        pgd_t * pgd;
        atomic_t mm_users;
        atomic_t mm_count;
        int map_count;
        struct rw_semaphore mmap_sem;
        spinlock_t page_table_lock;
        struct list_head mmlist;
        unsigned long start_code, end_code, start_data, end_data;
        unsigned long start_brk, brk, start_stack;
        unsigned long arg_start, arg_end, env_start, env_end;
        unsigned long rss, total_vm, locked_vm;
        unsigned long def_flags;
        unsigned long cpu_vm_mask;
        unsigned long swap_address;
        unsigned dumpable:1;
        mm_context_t context;
};
```

<include/linux/mm_types.h>



Memory Management/Virtual Memory

```
struct vm_area_struct {
        /* The first cache line has the info for VMA tree walking. */

        unsigned long vm_start;                 /* Our start address within vm_mm. */
        unsigned long vm_end;                   /* The first byte after our end address
                                                   within vm_mm. */

        /* linked list of VM areas per task, sorted by address */
        struct vm_area_struct *vm_next, *vm_prev;

        struct rb_node vm_rb;

        /*
         * Largest free memory gap in bytes to the left of this VMA.
         * Either between this VMA and vma->vm_prev, or between one of the
         * VMAs below us in the VMA rbtree and its ->vm_prev. This helps
         * get_unmapped_area find a free area of the right size.
         */
        unsigned long rb_subtree_gap;

        /* Second cache line starts here. */

        struct mm_struct *vm_mm;    /* The address space we belong to. */
        pgprot_t vm_page_prot;                  /* Access permissions of this VMA. */
        unsigned long vm_flags;                 /* Flags, see mm.h. */

        /*
         * For areas with an address space and backing store,
         * linkage into the address_space->i_mmap interval tree, or
         * linkage of vma in the address_space->i_mmap_nonlinear list.
         */
....
}
```

<include/linux/mm_types.h>

Memory Management/Virtual Memory

이름	내용
vm_mm	이 vm_area_sturct가 소속된 mm_struct
vm_start	이 영역이 시작 주소
vm_end	이 영역의 끝 주소 +1
vm_next	mm_struct로부터의 연결 리스트용
vm_page_prot	접근 권한
vm_flags	플래그
vm_rb	red-black(rb) 트리 연결용
shared	address_sapce 연결용
anon_vma_node	anon_vma 구조체 연결용
anon_vma	anon_vma 구조체
vm_ops	vm 연산
vm_pgoff	파일 오프셋
vm_file	맵하고 있는 파일
vm_private_data	개인용 데이터
vm_truncate_count	언맵(unmap) 시의 메모

<vm_area_struct>

Memory Management/Virtual Memory

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched.h>
static int mm1_init(void)
{
    struct task_struct *task;
    struct mm_struct *mm;

    for_each_process(task)
    {
        printk("task_id =%d ",task->pid);

        if(mm=(task->mm))
        {
            printk("start_code = %8lx ", mm->start_code);
            printk("end_code = %8lx ", mm->end_code);
            printk("start_data = %8lx ", mm->start_data);
            printk("end_data = %8lx\n", mm->end_data);
        }
        else
            printk("kernel task\n");
    }
    return 0;
}
static void mm1_exit(void)
{
    printk(KERN_ALERT "exit\n");
}
module_init(mm1_init);
module_exit(mm1_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kihan Choi");
```

<mm1.c>

Memory Management/Virtual Memory

```
1  O_TARGET := mm1.ko
2  obj-m := mm1.o
3
4  KERNEL_DIR := /lib/modules/$(shell uname -r)/build
5  MODULE_DIR := /lib/modules/$(shell uname -r)/kernel/mm1_module
6  PWD := $(shell pwd)
7
8
9  default :
10     $(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) modules
11
12 install :
13     mkdir -p $(MODULE_DIR)
14     cp -f $(O_TARGET) $(MODULE_DIR)
15
16 clean :
17     $(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) clean
18
19
```

<mm1 Makefile>

Memory Management/Virtual Memory

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched.h>
static int mm2_init(void)
{
    struct task_struct *task;
    struct mm_struct *mm;
    struct vm_area_struct *mmap;

    task = current;
    mm = task->mm;
    mmap = mm->mmap;

    printk("task id = %d\n", task->pid);

    do
    {
        printk("vm_start = %8lx ",mmap->vm_start);
        printk("vm_end   = %8lx ", mmap->vm_end);
        printk("vm_area  = %8lx\n", mmap->vm_end - mmap->vm_start);

    }while(mmap=(mmap->vm_next));

    return 0;
}
static void mm2_exit(void)
{
    printk(KERN_ALERT "exit\n");
}
module_init(mm2_init);
module_exit(mm2_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kihan Choi");
```

<mm1.c>

Memory Management/Virtual Memory

```
1  O_TARGET := mm2.ko
2  obj-m := mm2.o
3
4  KERNEL_DIR := /lib/modules/$(shell uname -r)/build
5  MODULE_DIR := /lib/modules/$(shell uname -r)/kernel/mm2_module
6  PWD := $(shell pwd)
7
8
9  default :
10     $(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) modules
11
12 install :
13     mkdir -p $(MODULE_DIR)
14     cp -f $(O_TARGET) $(MODULE_DIR)
15
16 clean :
17     $(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) clean
18
19
```

<mm2 Makefile>