

2019년 2학기 운영체제

Memory management

System Software Laboratory

School of Computer and Information Engineering

Kwangwoon Univ.

Contents

- 메모리 관리의 이해와 기법 소개
 - 물리 메모리와 가상 메모리
 - 메모리를 위한 자료 구조
 - 메모리 매핑
-
- 실습 1
 - 실습2

메모리 관리의 이해와 기법 소개

- 메모리 관리의 기본 핵심
 - 커널은 task를 언제, 어디에, 어떻게 물리 메모리에 적재할 지 관리해야 함
- 한정적인 물리 메모리의 해결 방안
 - 가상 메모리
 - 물리 메모리에는 수행에 필요한 부분만 적재

| 물리 메모리와 가상 메모리

■ 물리 메모리

- 시스템에 장착된 실제 메모리
- 메모리에 대한 실제 물리 주소를 가짐
- 0번지부터 장착된 메모리 크기까지의 범위

■ 가상메모리

- 실제 존재하지는 않지만 큰 메모리가 존재하는 것과 같은 효과
- 가상 주소는 물리 주소와 상관없이 각 task 마다 할당되는 논리적인 주소
- 리눅스에서는 **각 task 마다 256TB 가상 주소공간**을 할당
 - 각 task: 128TB, 커널영역: 128TB

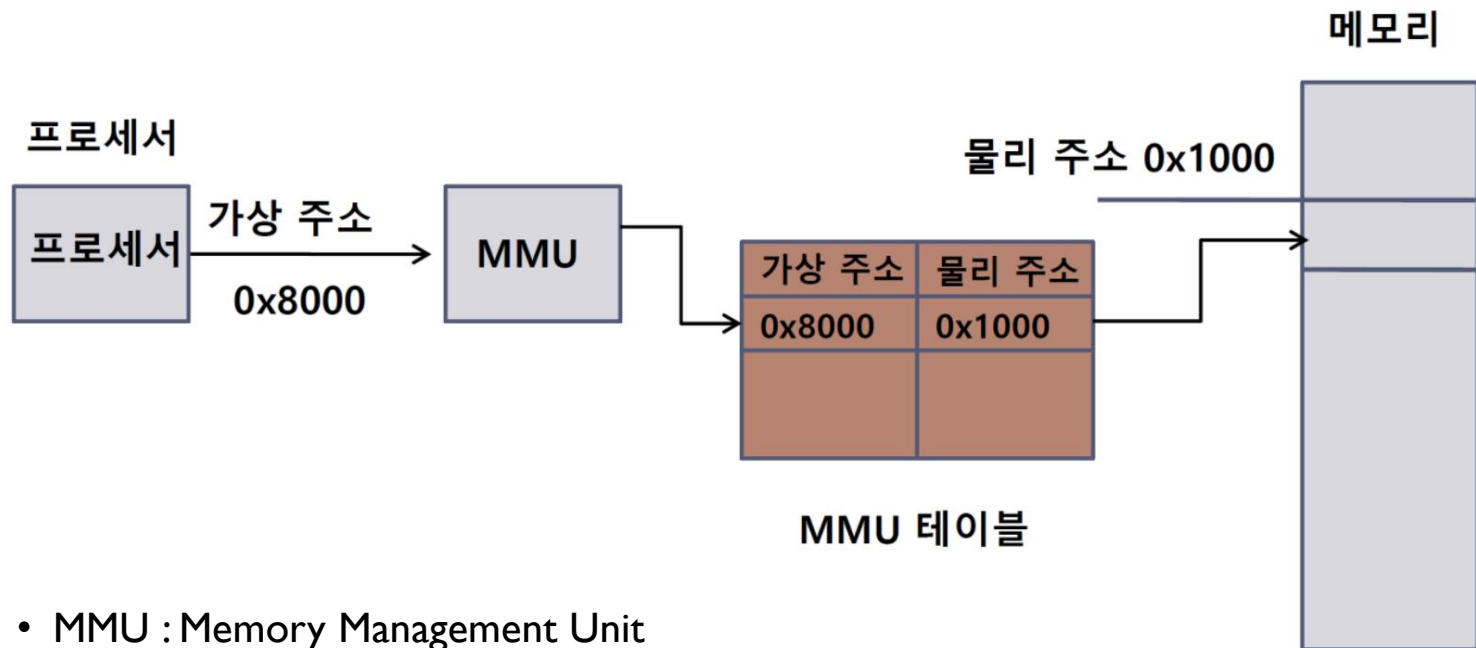
■ 가상주소를 물리주소로 변환하는 기법 필요

- Paging 기법을 사용

물리 메모리와 가상 메모리

가상 주소는

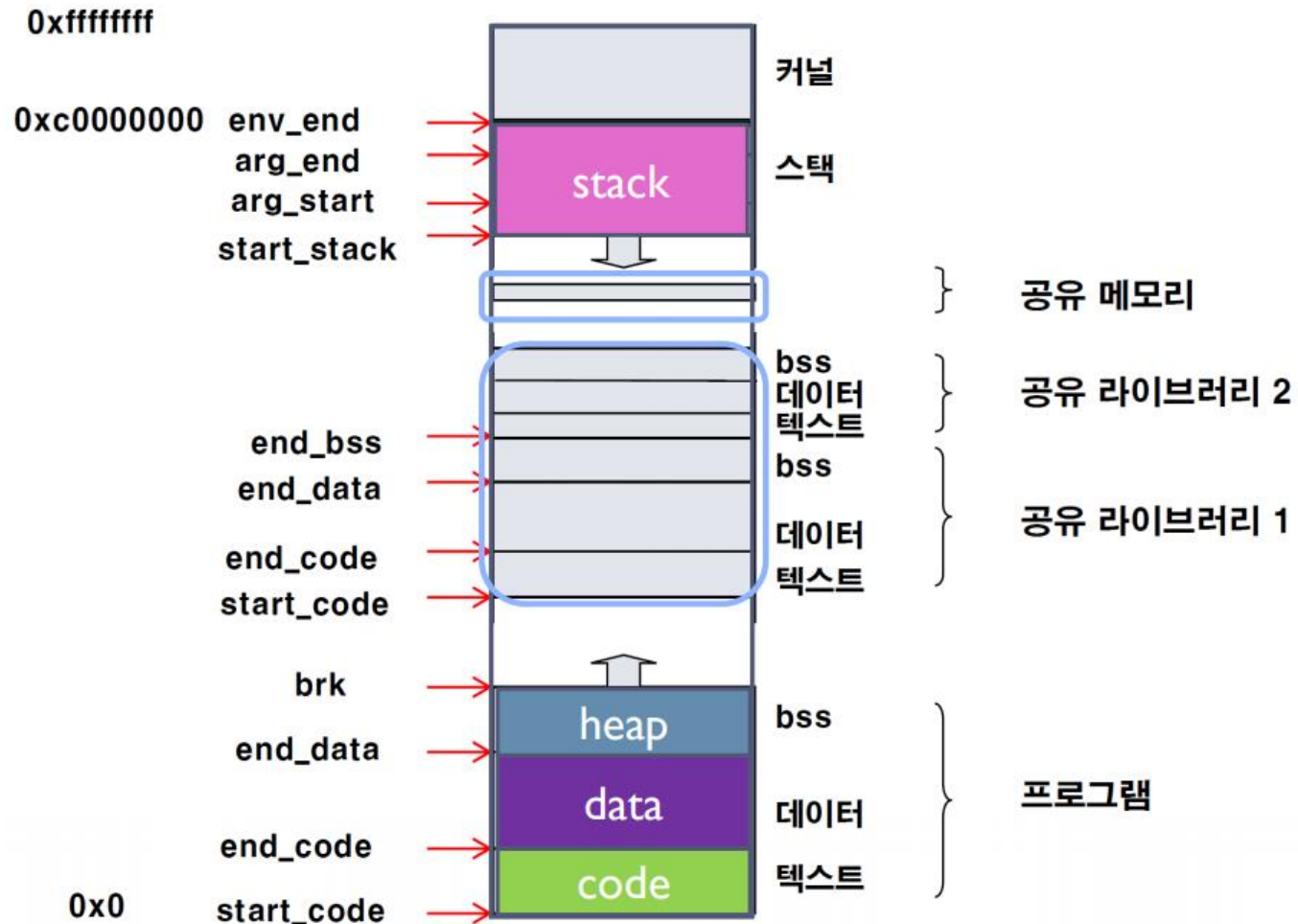
- 내부적으로 메모리 관리 기능을 통해 물리 주소로 변환
- 이를 실제 물리 메모리에 매핑



- MMU : Memory Management Unit

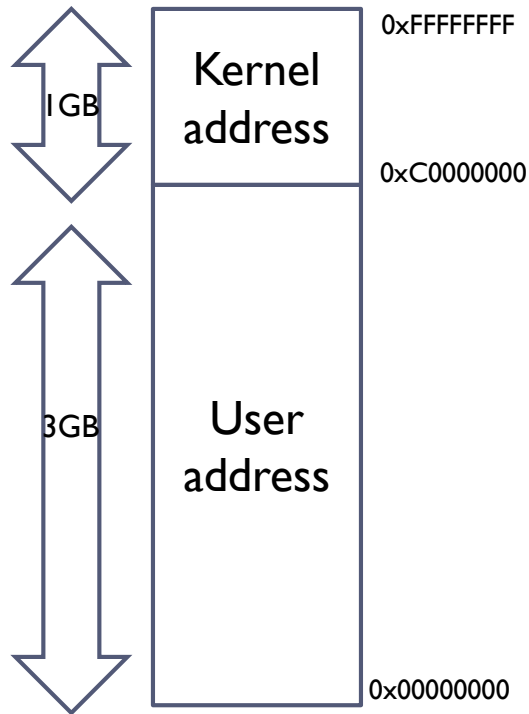
물리 메모리와 가상 메모리

- 리눅스 커널의 가상 메모리

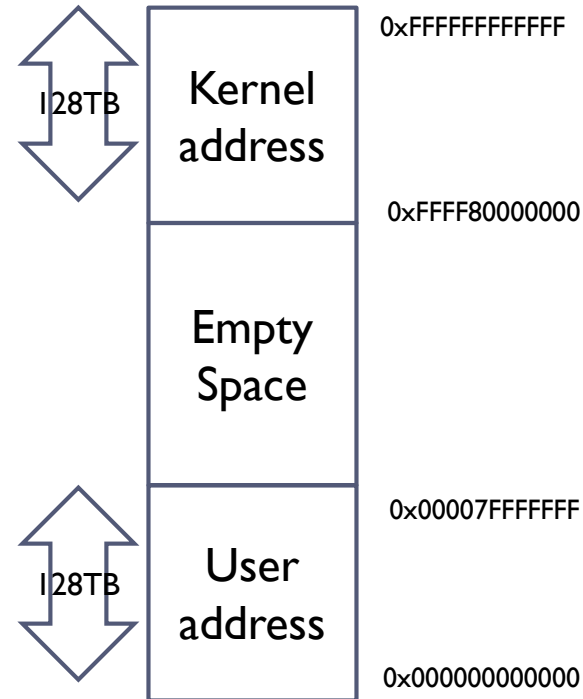


| 물리 메모리와 가상 메모리

- 리눅스 커널의 가상 메모리

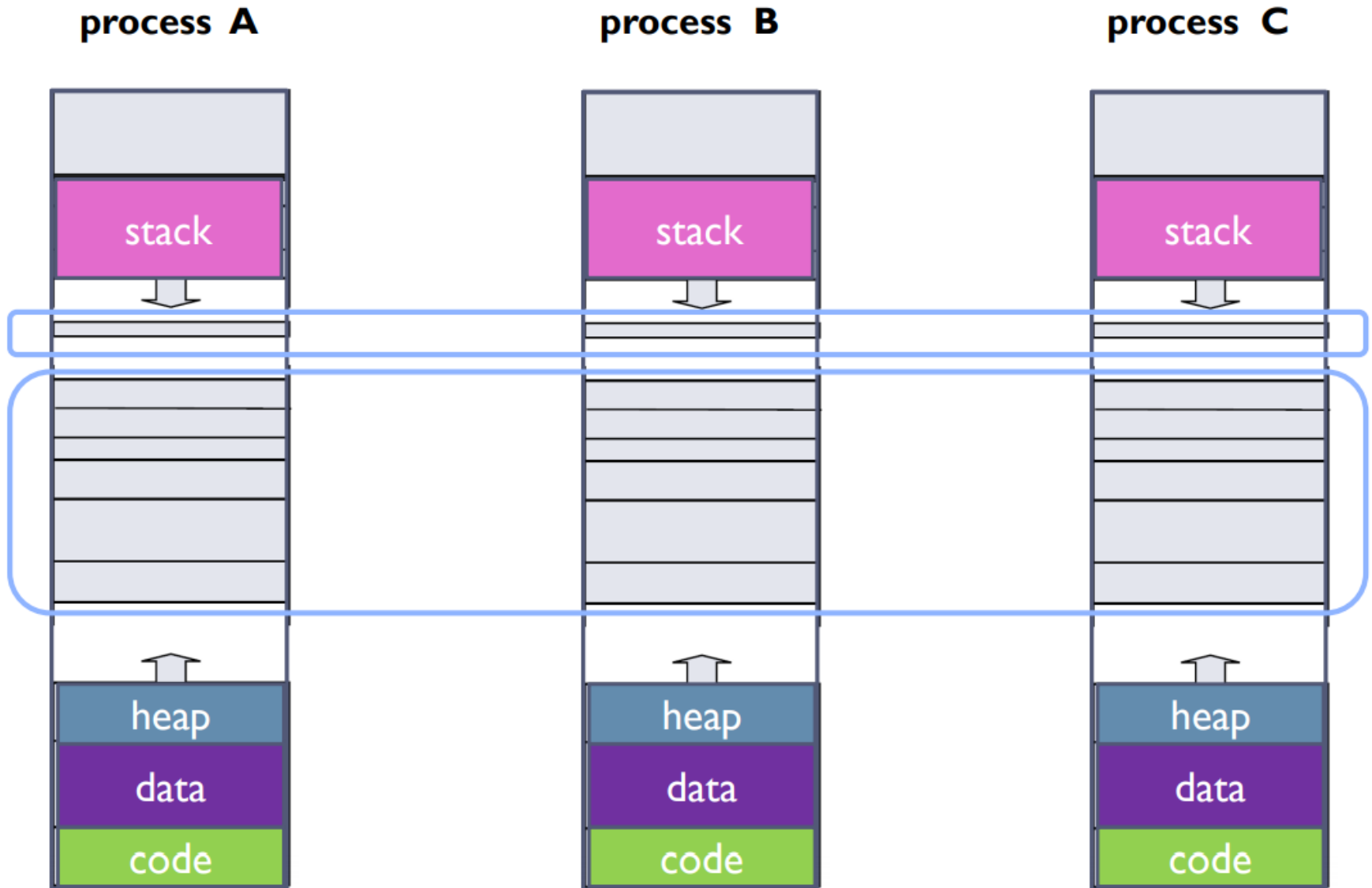


32bit 가상 메모리 공간

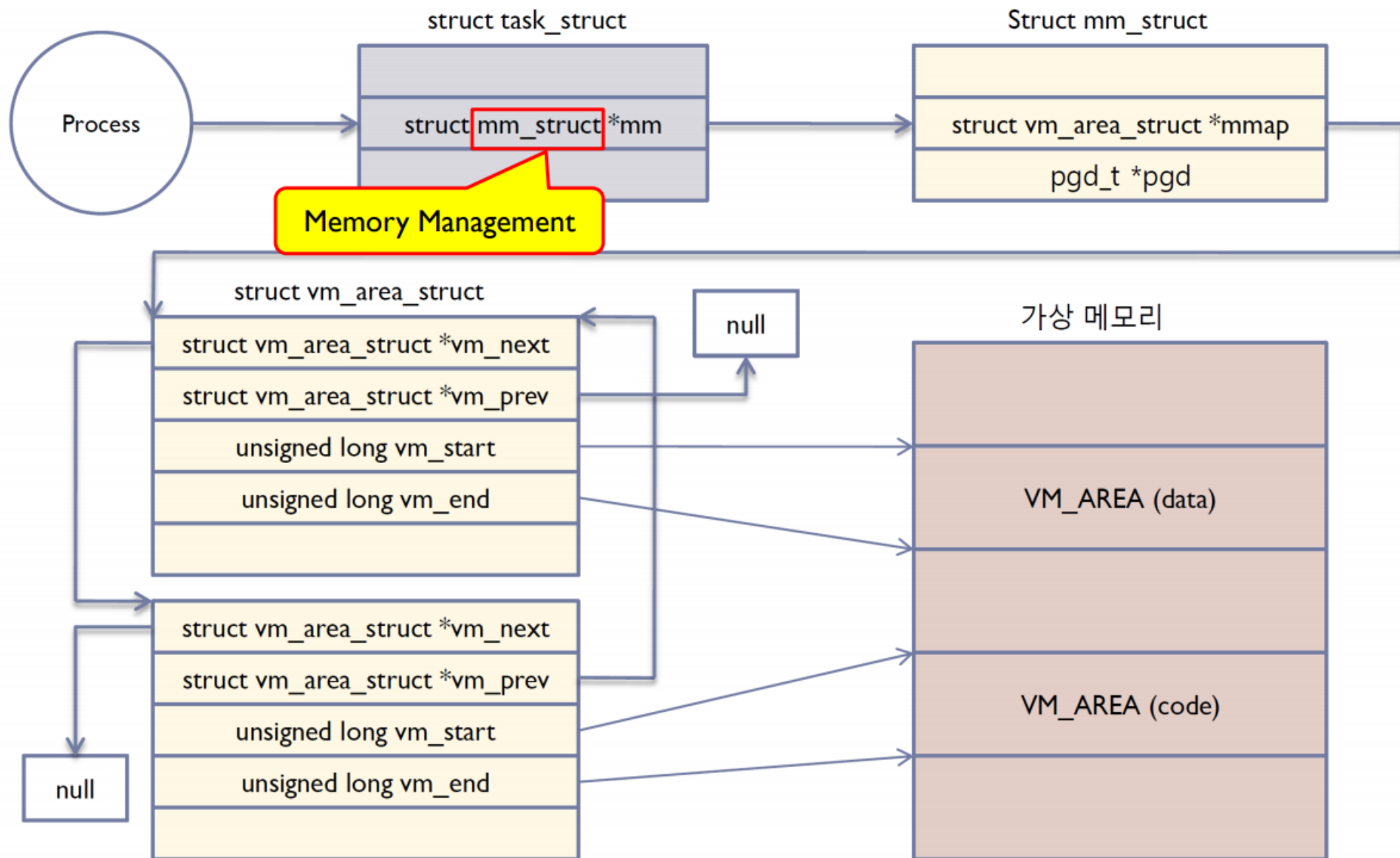


64bit 가상 메모리 공간

물리 메모리와 가상 메모리



메모리 관리를 위한 자료 구조



메모리 관리를 위한 자료 구조

- **mm_struct 주요 함수**
 - `struct mm_struct *get_task_mm(struct task_struct *task)`
 - use count를 1 증가
 - Task의 mm_struct를 가져옴
 - `void mmpu(struct mm_struct *mm)`
 - use count를 1 감소
 - use count가 0이 되면 할당된 메모리 공간을 해제

메모리 관리를 위한 자료 구조

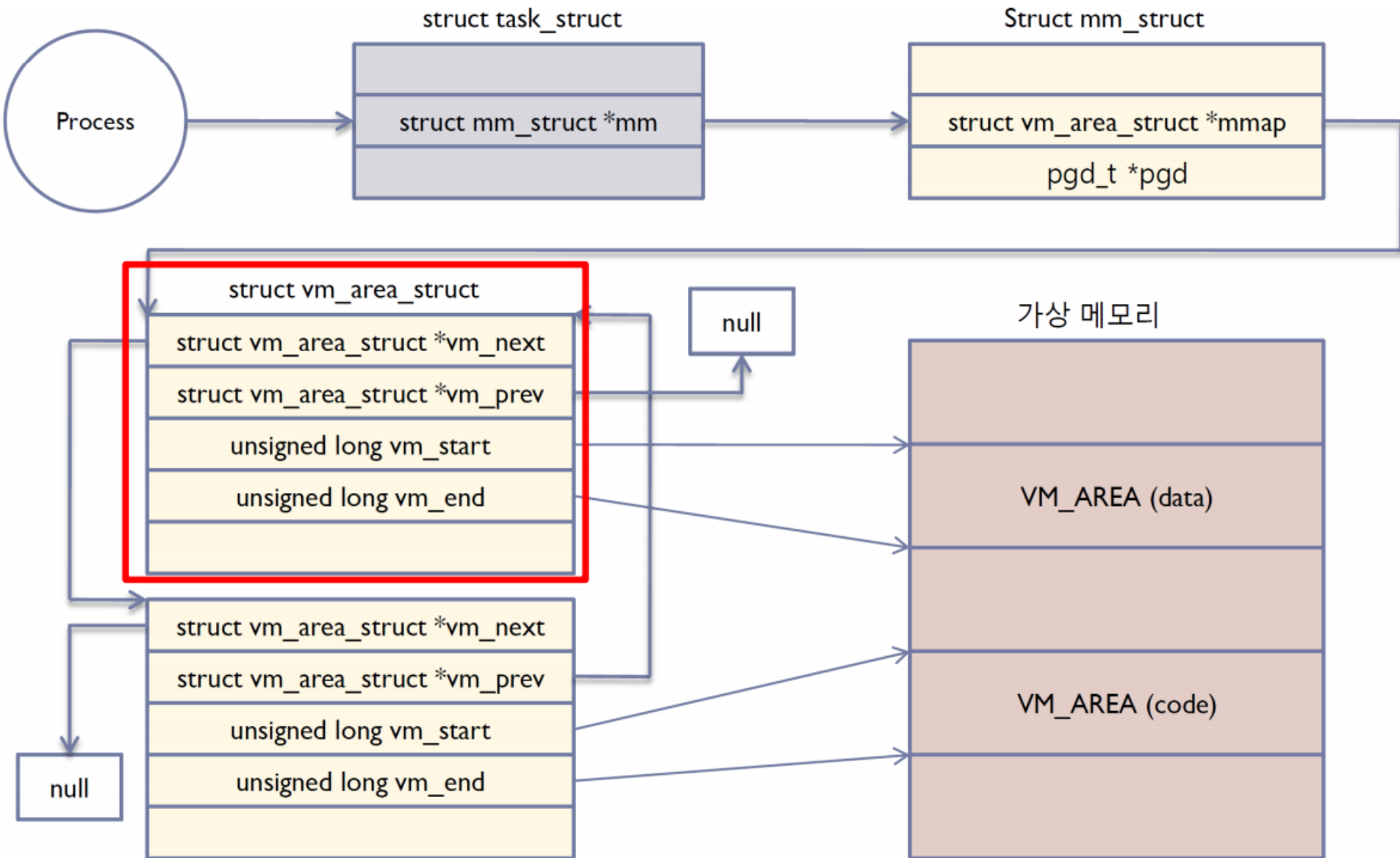
- **mm_struct 주요 멤버 변수**

- struct vm_area_struct *mmap
 - vm_area_struct list의 시작 주소
- pgd_t *pgd
 - 페이지 글로벌 디렉토리의 시작 주소
 - 페이지 글로벌 디렉토리
 - 가상 주소를 물리 주소로 변환하기 위한 최상위 테이블
- start_code, end_code
 - 코드 세그먼트 영역
 - 프로그램의 명령들이 들어가는 영역
- start_data, end_data
 - 데이터 세그먼트 영역
 - 프로그램에 선언된 전역 변수들로 구성
 - 초기화 된 변수가 들어가는 데이터 영역과 초기화 되지 않은 변수가 들어가는 BSS(Block Started by Symbol)로 구분

메모리 관리를 위한 자료 구조

- **mm_struct 주요 멤버 변수 (cont'd)**
 - start_brk, brk
 - 힙(heap)의 시작과 끝 주소를 가지는 변수
 - start_stack
 - 스택의 시작 위치
 - 함수 호출 시 전달되는 인자들과 복귀 주소 및 지역 변수로 구성
 - 스택의 메모리 할당 방향은 힙과 반대
 - total_vm
 - 할당된 전체 가상 메모리 크기
 - reserved_vm
 - 예약된 메모리 크기

메모리 관리를 위한 자료 구조



메모리 관리를 위한 자료 구조

- **vm_area_struct 주요 멤버 변수**

- vm_mm

- 이 가상 메모리 영역을 사용하고 있는 mm_struct 구조체를 가리키는 포인터 변수

- vm_start

- 영역의 시작 주소

- vm_end

- 영역의 끝 주소

- vm_next

- 다른 가상 메모리 블록을 가리키는 포인터 변수
 - 이를 따라가면 task가 사용하는 전체 가상 공간을 알 수 있음
 - 마지막 노드의 vm_next는 NULL 값을 가짐

| 메모리 매핑

- 파일 시스템의 파일과 메모리 공간을 매핑하는 방법
 - 파일에 대한 read/write 연산 마다,
 - 메모리 공간의 data를 업데이트 하고,
 - 저장장치의 내용을 업데이트 하는 것은 비효율적
 - 따라서 이를 **mapping 하여 처리**

메모리 매핑

- 관련 함수

```
void * mmap(void *addr, size_t length, int prot,  
            int flags, int fd, off_t offset);
```

- fd로 지정된 파일에서 offset을 시작으로 length 바이트 만큼을 addr로 대응시킴
 - 메모리에 파일이나 장치를 map
 - 호출한 프로세스의 가상 메모리 공간에 새로운 매핑을 생성
 - Parameters
 - addr : 가상 메모리 내 매핑하고자 하는 대략적인 위치 기록 (NULL: 커널이 할당)
 - length : 매핑 길이
 - prot : 페이지 권한
 - PROT_EXEC : 실행 권한
 - PROT_READ : 읽기 권한
 - PROT_WRITE : 쓰기 권한
 - PROT_NONE : 권한 없음
 - flags : 매핑 방식
 - MAP_SHARED : 같은 파일을 매핑한 다른 프로세스들에게 변경 사항 공유
 - MAP_PRIVATE : 변경 사항이 다른 프로세스들에게 공유되지 않음
 - ...
 - fd : 장치나 파일에 대한 file descriptor
 - offset : fd에 해당하는 장치/파일에서의 시작 위치 (PAGE_SIZE의 배수)

메모리 매핑

■ 관련 함수

```
int munmap (void *start, size_t length);
```

- 메모리에 매핑된 파일이나 장치를 unmap
- Parameters
 - start : 매핑 시작 주소
 - length : 매핑된 길이

```
int msync (void *start, size_t length, int flags);
```

- 메모리 매핑 후 변경된 사항을 파일에 반영 (동기화)
- Parameters
 - start : 매핑 시작 주소
 - Length : 매핑된 길이
 - Flags : 동기화 방식
 - MS_SYNC : 동기화를 요청하고, 동기화가 끝날 때 까지 대기
 - MS_ASYNC : 동기화를 요청하고 즉시 return
 - ...

| 메모리 매핑

- 관련 함수

```
int mprotect (void *addr, size_t length, int prot);
```

- 메모리 영역에 대한 접근 제어
- Parameters
 - addr : 시작 주소
 - length : 범위
 - prot : 제어 설정
 - PROT_NONE : 접근 불가능
 - PROT_READ : read 권한 부여
 - PROT_WRITE : write 권한 부여
 - PROT_EXEC : execute 권한 부여

실습 1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5 #include <sys/mman.h>
6 #include <assert.h>
7
8
9 void create_data(const char *filename);
10 void display_data(const char *filename);
11 void change_data(const char *filename);
12 void mmap_data(const char *filename);
13
14 int main(int argc, const char *argv[])
15 {
16     const char *filename;
17     assert(argc == 2);
18     filename = argv[1];
19     create_data(filename);
20     display_data(filename);
21     change_data(filename);
22     display_data(filename);
23     mmap_data(filename);
24     display_data(filename);
25
26     return EXIT_SUCCESS;
27 }
28
29 void create_data(const char *filename)
30 {
31     int fd;
32     int i;
33
34     fd = open(filename, O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
35     assert(fd != -1);
36     write(fd, "hello", 5);
37     close(fd);
38 }
39
40 void display_data(const char *filename)
41 {
42     int fd;
43     char data;
44     int i;
45
46
47     fd = open(filename, O_RDONLY);
48     assert(fd != -1);
49     for(i=0; i<5; ++i)
50     {
51         if(read(fd, &data, sizeof(char)) == sizeof(char))
52             printf("%c", data);
53     }
54     printf("\n");
55     close(fd);
56 }
57
58 void change_data(const char *filename)
59 {
60     int fd;
61     char data;
62
63     fd = open(filename, O_RDWR);
64     assert(fd != -1);
65     lseek(fd, 4*sizeof(char), SEEK_SET);
66     data = '!';
67     write(fd, &data, sizeof(char));
68     close(fd);
69 }
70
71 void mmap_data(const char *filename)
72 {
73     int fd;
74     char *map;
75     int pagesize;
76
77     fd = open(filename, O_RDWR);
78     pagesize = getpagesize();
79     map = mmap(0, pagesize, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
80     assert(map != MAP_FAILED);
81
82     map[0] &= ~0x20;
83     map[1] ^= 0x20;
84     map[2] &= 0xDF;
85     map[3] ^= ~0xDF;
86     msync(map, pagesize, MS_ASYNC);
87     munmap(map, pagesize);
88 }
```

실습 1

```
sslab@ubuntu:~/test_memory$ ./a.out test
hello
hell!
HELL!
sslab@ubuntu:~/test_memory$ cat test
HELL!sslab@ubuntu:~/test_memory$
```

실습 2

- PID를 입력 받고,
 - 이에 대한 프로세스의 이름과
 - 코드 영역의 시작 주소 값, 끝 주소 값을 가져오는 모듈 작성
- 결과 예시

```
root@ubuntu:/home/sslab/test_memory# ./test
root@ubuntu:/home/sslab/test_memory# dmesg
[ 1960.424388] name[test]
                        code[4194304~4196172]
```

- Test program

```
#include <unistd.h>
#include <sys/types.h>
#include <linux/unistd.h>

#define __NR_ftrace 336
int main(void){
    syscall(__NR_ftrace, getpid());
    return 0;
}
```

실습 2

- ftrace를 warpping 하여 구현

```
#include <linux/module.h>
#include <linux/kallsyms.h>
#include <linux/sched/mm.h>
#include <linux/syscalls.h>

#define __NR_ftrace 336

void **syscall_table;

asmlinkage pid_t (*real_ftrace)(struct pt_regs *regs);

__SYSCALL_DEFINEx(1, info, pid_t, pid)
{
    struct task_struct *t;
    struct mm_struct *mm;
    t = pid_task(find_vpid(pid), PIDTYPE_PID);
    mm = get_task_mm(t);

    printk(KERN_INFO "name[%s]\ncode[%d~%d]\n", t->comm, mm->start_code, mm->end_code);
    mput(mm);
}
```