Physical Page Map Analysis

1.1 os_pagemap proc 생성

```
static ssize_t ospagemap_read(struct file *file, char __user *buf, size_t size, loff_t *ppos)
{
        1.3에 기술
}
static int ospagemap_open(struct inode *inode, struct file *file)
{
        printk("ospagemap opened");
        return 0;
}
const struct file_operations proc_ospagemap_operations = {
        .llseek = mem_lseek,
        .read = ospagemap_read,
        .open = ospagemap_open,
};
static int __init proc_ospagemap_init(void)
{
        proc_create("os_pagemap", 0, NULL, &proc_ospagemap_operations);
        return 0;
}
fs_initcall(proc_ospagemap_init);
```

1.2 task 정보 구조체

```
struct task_info {
        char pname[100];    <- 읽어온 task의 이름을 저장할 변수
        int pid;            <- 읽어온 task의 pid를 저장할 변수
};
```

1.3 ospagemap_read - 프로세스 정보 읽어오기

```
static ssize_t ospagemap_read(struct file *file, char __user *buf, size_t size, loff_t *ppos)
{
        int i;
        const int BUF_SIZE = 1000000;   <- 읽어온 데이터를 저장할 버퍼의 크기
        const int INFO_SIZE = 500;      <- task_info 구조체의 개수
        char *pagemap_buf = kmalloc(BUF_SIZE, GFP_KERNEL);    <- 읽어온 데이터를 저장할 버퍼
        int cur = 0, bytes = 0, num_of_process = 0;
        struct task_struct *p = NULL;
        struct task_info *info = kmalloc(INFO_SIZE, GFP_KERNEL);   <- task의 이름과 pid를 읽어와서 저장할 구조체

        read_lock(&tasklist_lock);    <- 프로세스에 접근하기 위해 read lock 설정
        for_each_process(p) {         <- 모든 프로세스를 루프
                if (NULL == p) {    <- null이 나오면 중지
                        break;
                }

                (info + num_of_process)->pid = p->pid;                      <- task에서 pid 읽어오고 저장
                sprintf((info + num_of_process)->pname, "%s", p->comm);     <- task에서 이름 읽어오고 저장

                num_of_process++;    <- 프로세스 개수 1 증가
        }
```

```
        read_unlock(&tasklist_lock);     <- 프로세스를 모두 읽었기 때문에 read lock 해제

        위에서 읽은 프로세스의 개수만큼 loop
        for (i = 0; i < num_of_process; i++) {
                struct pid *pid_struct;             <- 찾은 task의 pid 구조체를 가리키는 포인터 변수
                struct task_struct *task = NULL;    <- 찾은 task를 가리키는 포인터 변수
                struct mm_struct *mm;               <- 찾은 task의 mm을 가리키는 포인터 변수
                struct vm_area_struct *vma;         <- mm을 loop하기 위한 포인터 변수

                pid_struct = find_get_pid((info + i)->pid);    <- 이전 loop에서 저장한 pid를 통해 pid_struct 찾음
                task = pid_task(pid_struct, PIDTYPE_PID);      <- pid_struct를 통해 task 발견

                if (NULL == task || !task) {      <- 발견한 task가 이미 존재하지 않으면 넘어감
                        continue;
                }

                mm = task->mm;
                if (NULL != mm) {
                        down_read(&mm->mmap_sem);   <- mm에 접근하기 위해 세마포어 설정

                        mm->mmap에 연결되어 있는 모든 영역 loop
                        for (vma = mm->mmap; vma != NULL; vma = vma->vm_next) {
                                가상주소를 통해 물리주소를 구하고 4096으로 나눠서 ppn을 구함
                                unsigned int ppn = virt_to_phys((void *)vma->vm_start) / 4096;
                                unsigned long vpn = vma->vm_start / 4096;

                                int segment_type = -1;
                                char segment_name[4][6] = { "HEAP", "STACK", "DATA", "*" };
                                char fpath[200];       <- 파일 경로를 저장할 변수

                                fpath[0] = 0;
                                memory segment 타입과 이름을 읽어옴
                                segment_type = get_segment_name(vma, task->pid, fpath);

                                읽어온 데이터와, VMA의 이름이 존재하면 출력하고 없으면 무슨 segment인지 출력
                                bytes = snprintf((cur > 0) ? cur + pagemap_buf : pagemap_buf, BUF_SIZE - cur,
                                         "Virt %lu Phy %u VMA %s PID %d PNAME %s\n", vpn, ppn,
                                          ((segment_type == -1)? fpath : segment_name[segment_type]),
                                          task->pid, task->comm);
                                cur += bytes;
                        }
                        up_read(&mm->mmap_sem);         <- mm 사용을 다 했으므로 세마포어 해제
                }
        }
        copy_to_user(buf, pagemap_buf, (size < cur) ? size : cur);   <- 읽어온 데이터를 user 버퍼에 전달

        kfree((void*)pagemap_buf);    <- 사용 다 했으므로 메모리 해제

        return (size < cur) ? size : cur;
}
```

1.4 VMA 이름 읽어오기

```
static int is_stack(struct vm_area_struct *vma, int is_pid)
{
        int stack = 0;
        if (is_pid) {
                VMA의 주소와 stack의 주소값을 비교해서 segment가 stack에 있는지 판단
                stack = vma->vm_start <= vma->vm_mm->start_stack && vma->vm_end >= vma->vm_mm->start_stack;
```

```
        }
        return stack;
}
int get_segment_name(struct vm_area_struct *vma, int is_pid, char fpath[]) {
        struct mm_struct *mm = vma->vm_mm;
        struct file *file = vma->vm_file;
        int len, i;

        char *tmp;
        char *pathname;
        struct path *path;

        VMA에 속한 파일이 존재할 경우 이름을 구함
        if (file) {
                spin_lock(&vma->vm_file->f_lock);     <- 파일을 읽기 위해 f_lock의 스핀락 설정
                if (!file) {
                        spin_unlock(&vma->vm_file->f_lock);
                        return -ENOENT;
                }

                path = &file->f_path;
                path_get(path);           <- path_get 함수를 통해 읽어옴

                spin_unlock(&vma->vm_file->f_lock);    <- 파일 사용을 마쳤으므로 스핀락 해제

                tmp = (char *)__get_free_page(GFP_TEMPORARY);

                if (!tmp) {
                        path_put(path);
                        return 3;          <- 빈칸
                }

                pathname = d_path(path, tmp, PAGE_SIZE);    <- d_path 함수를 통해 전체 경로를 가져옴
                path_put(path);

                if (IS_ERR(pathname)) {
                        free_page((unsigned long)tmp);
                        return PTR_ERR(pathname);
                }

                얻어온 파일경로를 저장
                len = strlen(pathname);
                for (i = 0; i <= len; i++) {
                        fpath[i] = pathname[i];
                }
                free_page((unsigned long)tmp);

                return -1;     <- VMA 파일명이 존재한다는 뜻
        }

        VMA 파일명이 존재하지 않을 경우
        주소의 시작과 끝을 통해 힙에 속해있는지 판단
        if (vma->vm_start <= mm->brk && vma->vm_end >= mm->start_brk) {
                return 0;
                // heap
        }
         주소의 시작과 끝을 통해 data에 속해있는지 판단
        if (vma->vm_start <= mm->start_data && vma->vm_end <= mm->end_data) {
                return 2;
```

```
                // data
        }
         주소의 시작과 끝을 통해 스텍에 속해있는지 판단
        if (is_stack(vma, is_pid)) {
                return 1;
                // stack
        }
        return 3; // blank          <- 아무데도 속하지 않음
}
```

1.5.1 Parsing 방법

읽어온 데이터는 전부 String으로 처리를 했다.

```
bytes = snprintf((cur > 0) ? cur + pagemap_buf : pagemap_buf, BUF_SIZE - cur, "Virt %lu\tPhy %u\t VMA %s\t PID %d\t PNAME %s\n", vpn, ppn, ((segment_type == -1)? fpath : segment_name[segment_type]), task->pid, task->comm);
```

위와 같이 데이터마다 tap('\t')으로 구분을 지어놓았고, 다음 데이터는 개행('\n')을 했기 때문에 user 에서 os_pagemap.txt를 작성할 때 특별히 parsing을 할 필요는 없다.

1.5.2 os_pagemap.txt 생성

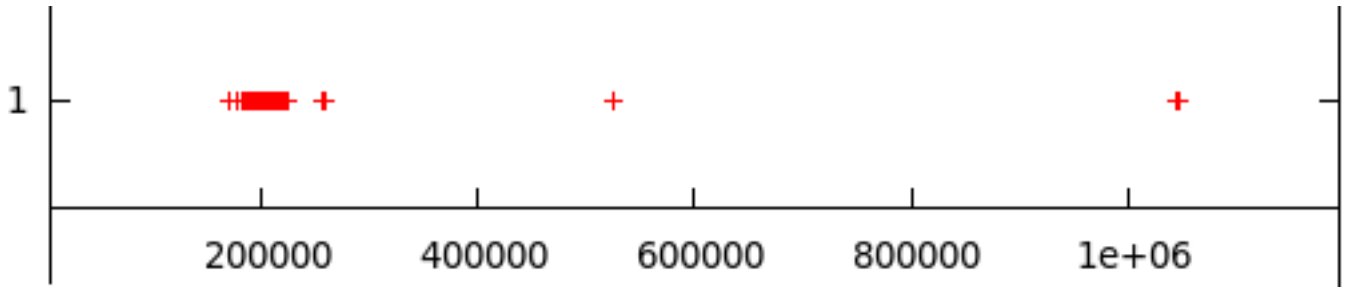
```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main()
{
        FILE *write = fopen("os_pagemap.txt", "w");      <- os_pagemap.txt를 생성하기 위한 파일포인터
        const int SIZE = 4 * 1024 * 1024;                <- kernel에서 최대 4MB까지 읽어올 수 있으므로 4MB로 설정
        char buf[SIZE];       <- 읽어온 데이터 String을 저장할 변수
        int fd = open("/proc/os_pagemap", O_RDONLY);     <- kernel에서 생성한 os_pagemap proc module 사용
        int bytes = read(fd, buf, SIZE);                 <- 읽어온 데이터를 buf에 저장

        fprintf(write, "%s", buf);    <- os_pagemap.txt에 buf를 write함
        fclose(write);

        return 0;
}
```

2.1 Page Map Graph



2.2 특정 프로세스 VMA 메모리 사용량

이름	PID	VPN	PFN	VMA
cron	1929	16	524304	/usr/sbin/cron
cron	1929	38	524326	/usr/sbin/cron
cron	1929	39	524327	/usr/sbin/cron
cron	1929	40	524328	HEAP
cron	1929	748293	224005	/lib/arm-linux-gnueabihf/libnss_files-2.21.so
cron	1929	748300	224012	/lib/arm-linux-gnueabihf/libnss_files-2.21.so
cron	1929	748315	224027	/lib/arm-linux-gnueabihf/libnss_files-2.21.so
cron	1929	748316	224028	/lib/arm-linux-gnueabihf/libnss_files-2.21.so
cron	1929	748317	224029	/lib/arm-linux-gnueabihf/libnss_nis-2.21.so
cron	1929	748324	224036	/lib/arm-linux-gnueabihf/libnss_nis-2.21.so
cron	1929	748339	224051	/lib/arm-linux-gnueabihf/libnss_nis-2.21.so
cron	1929	748340	224052	/lib/arm-linux-gnueabihf/libnss_nis-2.21.so
cron	1929	748341	224053	/lib/arm-linux-gnueabihf/libnsl-2.21.so
cron	1929	748354	224066	/lib/arm-linux-gnueabihf/libnsl-2.21.so
cron	1929	748369	224081	/lib/arm-linux-gnueabihf/libnsl-2.21.so
cron	1929	748370	224082	/lib/arm-linux-gnueabihf/libnsl-2.21.so
cron	1929	748371	224083	*
cron	1929	748373	224085	/lib/arm-linux-gnueabihf/libnss_compat-2.21.so
cron	1929	748378	224090	/lib/arm-linux-gnueabihf/libnss_compat-2.21.so
cron	1929	748393	224105	/lib/arm-linux-gnueabihf/libnss_compat-2.21.so
cron	1929	748394	224106	/lib/arm-linux-gnueabihf/libnss_compat-2.21.so
cron	1929	748395	224107	/usr/lib/locale/locale-archive
cron	1929	748907	224619	/lib/arm-linux-gnueabihf/libpthread-2.21.so
cron	1929	748923	224635	/lib/arm-linux-gnueabihf/libpthread-2.21.so
cron	1929	748924	224636	/lib/arm-linux-gnueabihf/libpthread-2.21.so
cron	1929	748925	224637	*
cron	1929	748927	224639	/lib/arm-linux-gnueabihf/libpcre.so.3.13.1
cron	1929	749001	224713	/lib/arm-linux-gnueabihf/libpcre.so.3.13.1
cron	1929	749016	224728	/lib/arm-linux-gnueabihf/libpcre.so.3.13.1
cron	1929	749017	224729	/lib/arm-linux-gnueabihf/libpcre.so.3.13.1
cron	1929	749018	224730	/lib/arm-linux-gnueabihf/libdl-2.21.so
cron	1929	749020	224732	/lib/arm-linux-gnueabihf/libdl-2.21.so

cron	1929	749035	224747	/lib/arm-linux-gnueabihf/libdl-2.21.so
cron	1929	749036	224748	/lib/arm-linux-gnueabihf/libdl-2.21.so
cron	1929	749037	224749	/lib/arm-linux-gnueabihf/libaudit.so.1.0.0
cron	1929	749059	224771	/lib/arm-linux-gnueabihf/libaudit.so.1.0.0
cron	1929	749074	224786	/lib/arm-linux-gnueabihf/libaudit.so.1.0.0
cron	1929	749075	224787	/lib/arm-linux-gnueabihf/libaudit.so.1.0.0
cron	1929	749076	224788	*
cron	1929	749086	224798	/lib/arm-linux-gnueabihf/libc-2.21.so
cron	1929	749303	225015	/lib/arm-linux-gnueabihf/libc-2.21.so
cron	1929	749318	225030	/lib/arm-linux-gnueabihf/libc-2.21.so
cron	1929	749320	225032	/lib/arm-linux-gnueabihf/libc-2.21.so
cron	1929	749321	225033	*
cron	1929	749324	225036	/lib/arm-linux-gnueabihf/libselinux.so.1
cron	1929	749345	225057	/lib/arm-linux-gnueabihf/libselinux.so.1
cron	1929	749360	225072	/lib/arm-linux-gnueabihf/libselinux.so.1
cron	1929	749361	225073	/lib/arm-linux-gnueabihf/libselinux.so.1
cron	1929	749362	225074	*
cron	1929	749363	225075	/lib/arm-linux-gnueabihf/libpam.so.0.83.1
cron	1929	749371	225083	/lib/arm-linux-gnueabihf/libpam.so.0.83.1
cron	1929	749387	225099	/lib/arm-linux-gnueabihf/libpam.so.0.83.1
cron	1929	749388	225100	/lib/arm-linux-gnueabihf/libpam.so.0.83.1
cron	1929	749413	225125	/lib/arm-linux-gnueabihf/ld-2.21.so
cron	1929	749445	225157	/usr/lib/locale/locale-archive
cron	1929	749446	225158	*
cron	1929	749452	225164	/lib/arm-linux-gnueabihf/ld-2.21.so
cron	1929	749453	225165	/lib/arm-linux-gnueabihf/ld-2.21.so
cron	1929	780275	255987	STACK
cron	1929	782270	257982	*
cron	1929	782271	257983	*
cron	1929	782272	257984	*

2.3 최다 사용 공유 데이터

/lib/arm-linux-gnueabihf/libc-2.21.so 로 Mapping 된 Data 가 총 284 번의 횟수 기록으로 최다 사용