CS353 Linux 内核 Project 3

第一部分

实验要求

编写一个内核模块，用户可以通过 proc 文件系统与模块通信，读写任意进程的内存空间。

具体来说，该模块需要进行以下操作：

1. 创建名为 `mtest` 的 proc 文件。
2. 从 proc 文件中读取用户指令，指令形式为 `r <pid> <address>` 或者是 `w <pid> <address> <content>`。`pid` 为目标进程的进程号。`address` 为 16 进制的目标虚拟地址。`r` 表示读取该进程该地址的一字节内容，`w` 表示向该进程的该地址写入一字节 `content` 的内容。
3. 取得进程的 `task_struct`，并根据其找到目的地址对应的页，并得到页对应的 `struct page` 结构体。`pfn_to_page` 宏可以从页框号得到对应的 `struct page` 结构体。
4. 因为模块代码处于内核内存空间中，所以并不能直接访问该页的内容，还需要将该页映射到内核内存空间中，内核函数 `kmap_local_page` 可以完成这项工作。
5. 读取或者修改对应的内存地址并存储下来。
6. 若用户指令为读取指令，用户必须通过读取 proc 文件，得到对应的值。

预期结果

这里提供一个测试程序：

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

int main(void)
{
    char c = 42;
    pid_t pid = getpid();
    printf("pid: %d\n", pid);
    printf("addr: %lx\n", (unsigned long)(&c));
    printf("content: %d\n", (int)c);

    FILE* fp = fopen("/proc/mtest", "w");
    fprintf(fp, "r %d %lx\n", pid, (unsigned long)(&c));
    fclose(fp);
    fp = fopen("/proc/mtest", "r");
    int r;
    fscanf(fp, "%d", &r);
    printf("read result: %d\n", r);
    fclose(fp);

    fp = fopen("/proc/mtest", "w");
    fprintf(fp, "w %d %lx 1\n", pid, (unsigned long)(&c));
```

```
    fclose(fp);
    printf("content: %d\n", (int)c);
    return 0;
}
```

预期得到类似以下的结果：

```
pid: 38449
addr: 7ffd8d3c9c97
content: 42
read result: 42
content: 1
```

第二部分

实验要求

编写一个内核模块，创建一个 proc 文件，在用户使用 `mmap` 系统调用时，为用户设置好对应的映射。

具体来说，该模块需要完成以下操作：

1. 创建 proc 文件，同时设置好该文件被调用 `mmap` 系统调用时的回调函数。这个可以通过设置 `struct proc_ops` 结构体的 `proc_mmap` 成员实现。
2. 通过 `alloc_page` 函数分配一个物理页，并向物理页中写入一些特殊内容，方便和其他页做区分。在写入内容之前，和第一部分的模块一样，也要将该物理页映射到内核内存空间中。
3. 当 proc 文件的 `proc_mmap` 回调函数被调用时，利用 `remap_pfn_range` 函数将之前所分配的页与用户内存空间对应起来。
4. 用户可以直接访问 `mmap` 映射得到的内存空间并读到写入的特殊内容。

预期结果

这里提供一个测试程序：

```
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>

#define SIZE 1 << 12

int main(void)
{
    int fd = open("/proc/maptest", O_RDONLY);
    if (fd == -1) {
        perror("open /proc/maptest error\n");
    }
    char* buf = (char*) mmap(NULL, SIZE, PROT_READ, MAP_PRIVATE, fd, 0);
    fputs(buf, stdout);
    munmap(buf, SIZE);
```

```
    return 0;
}
```

预期得到类似以下的结果：

```
Listen to me say thanks
Thanks to you, I'm warm all the time
I thank you
For being there
The world is sweeter
I want to say thanks
Thanks to you, love is in my heart
I thank you, for being there
To bring happiness
```

提示

1. 本次 project 可能用到一些课堂上没有讲过的函数，可以通过网站 https://elixir.bootlin.com/linux/latest/source 搜索对应的函数名，在搜索结果中选择 **Documented in xx files:** 的结果，阅读注释来了解该函数以及各个参数的作用。
2. 要使用 `struct proc_ops` 结构体，**内核版本要在5.6 以上**。
3. 可以从模版代码出发： https://github.com/chengjiagan/CS353-2022-Spring。本次 project 的代码在目录 `projec3` 下，测试程序的代码也在其中。
4. 第二部分的模块中需要写入的特殊内容可以自由发挥，只要有区分性即可。模版代码中的仅供参考。