**添加系统调用**

**一、题目**

添加一个系统调用（相关知识点参考实验一章的“系统调用添加”）, 该系统调用接受两个参数：参数1：以整型数表示的自己学号的后3位；参数2：flag，取值为0或1，若为0，

该系统调用的返回值为参数1的个位。若为1。该系统调用的返回值为参数1的十位。

此外，加入内核互斥锁，使得两个进程在调用该系统调用时，能够做到互斥访问该系统调用。

i.自己所添加的系统调用的位置和修改点，以及为什么在这些位置上进行修改。

ii.自己系统调用关键语句的含义。

iii. 如何编译内核并调用自己的系统调用。

**二、位置和修改点**

1.定义系统调用号

文件：arch/x86/entry/syscalls/syscall\_64.tbl

修改点：文件末尾添加

548 公共 peep\_page sys\_peep\_page

原因：这里定义了系统调用的编号（548），名称（peep\_page），以及内核中对应的系统调用处理函数（sys\_peep\_page）。

2. 声明系统调用

文件：include/linux/syscalls.h

修改点：文件末尾的#endif前添加

asmlinkage long sys\_peep\_page(pid\_t tar\_pid\_nr, unsigned long tar\_addr, unsigned long my\_addr);

原因：这里声明了系统调用函数sys\_peep\_page，使得内核其他部分知道这个系统调用的存在。

3.定义系统调用号

文件：include/uapi/asm-generic/unistd.h

修改点：

#定义 \_\_NR\_peep\_page 548

\_\_SYSCALL(\_\_NR\_peep\_page， sys\_peep\_page)

#定义 \_\_NR\_peep\_page 548

\_\_SYSCALL(\_\_NR\_peep\_page， sys\_peep\_page)

原因：定义系统调用号548，并关联到系统调用函数sys\_peep\_page。

4.实现系统调用

文件：mmap.c

修改点：在文件中添加系统调用的实现

1. *// 定义互斥锁*
2. static DEFINE\_MUTEX(peep\_page\_mutex);
3. static pte\_t\*
4. addr\_to\_pte(struct mm\_struct \*mm, unsigned long addr)
5. {
6. return pte\_offset\_kernel(pmd\_off(mm, addr), addr);
7. }
8. SYSCALL\_DEFINE3(peep\_page, pid\_t, tar\_pid\_nr, unsigned long tar\_addr, unsigned long my\_addr)
9. {
10. int flag = (int)tar\_addr;  *// 使用 tar\_addr 作为 flag 参数*
11. int param1 = (int)my\_addr; *// 使用 my\_addr 作为 param1 参数*
12. int result;
13. *// 获取互斥锁*
14. mutex\_lock(&peep\_page\_mutex);
15. *// 根据 flag 返回相应的值*
16. if (flag == 0) {
17. result = param1 % 10;  *// 返回个位*
18. } else if (flag == 1) {
19. result = (param1 / 10) % 10;  *// 返回十位*
20. } else {
21. *// 设置错误码并返回*
22. result = -EINVAL; //会返回-1
23. mutex\_unlock(&peep\_page\_mutex);
24. return result;
25. }
26. *// 释放互斥锁*
27. mutex\_unlock(&peep\_page\_mutex);
28. return result;
29. }

原因：实现了系统调用的核心逻辑，并使用互斥锁保证系统调用的互斥访问。

**三、关键语句的含义**

1. 互斥锁定义和初始化

static DEFINE\_MUTEX(peep\_page\_mutex);

含义：定义并初始化一个互斥锁，用于保护系统调用的互斥访问。

1. 系统调用定义

SYSCALL\_DEFINE3(peep\_page, pid\_t, tar\_pid\_nr, unsigned long tar\_addr, unsigned long my\_addr)

含义：定义一个新的系统调用peep\_page，接受三个参数：tar\_pid\_nr、tar\_addr、my\_addr。

1. 参数解析和日志打印
2. int flag = (int)tar\_addr;  *// 使用 tar\_addr 作为 flag 参数*
3. int param1 = (int)my\_addr; *// 使用 my\_addr 作为 param1 参数*
4. printk("peep\_page: running! param1: %d, flag: %d\n", param1, flag);

含义：解析传入的参数，将tar\_addr解释为flag，将my\_addr解释为param1，并打印日志以便调试。

1. 获取互斥锁

mutex\_lock(&peep\_page\_mutex);

含义：在进入关键区域前获取互斥锁，保证系统调用的互斥访问。

1. 根据flag返回相应值
2. if (flag == 0) {
3. result = param1 % 10;  *// 返回个位*
4. } else if (flag == 1) {
5. result = (param1 / 10) % 10;  *// 返回十位*
6. } else {
7. *// 设置错误码并返回*
8. result = -EINVAL;
9. mutex\_unlock(&peep\_page\_mutex);
10. return result;
11. }

含义：根据flag的值，返回param1的个位或十位。如果flag值非法，则返回错误码-EINVAL。

**-flag参数传递的详细说明**

在fine.c中，调用了自定义的系统调用peep\_page，该系统调用需要两个参数：一个表示学号后3位的整型数param1，一个表示操作标志的整型数flag。具体的传参过程如下：

**1）在用户空间调用系统调用**

int result0 = peep\_page(param1, flag);

在fine.c中，peep\_page函数被调用并传入两个参数param1和flag。其中，param1是整型数894，flag是整型数0或1。

**2）peep\_page函数定义**

1. static inline long peep\_page(int param1, int flag)
2. {
3. return syscall(\_\_NR\_peep\_page, param1, flag);
4. }

该函数内部使用syscall函数来调用系统调用peep\_page，\_\_NR\_peep\_page是系统调用号548。参数param1和flag被传递给内核空间的系统调用处理函数。

**3) 内核空间系统调用处理函数**

SYSCALL\_DEFINE3(peep\_page, pid\_t, tar\_pid\_nr, unsigned long tar\_addr, unsigned long my\_addr)

在内核中，系统调用处理函数sys\_peep\_page接收三个参数：tar\_pid\_nr、tar\_addr、my\_addr。在用户空间传入的param1和flag分别对应于my\_addr和tar\_addr。通过参数转换：

1. int flag = (int)tar\_addr;  *// 使用 tar\_addr 作为 flag 参数*
2. int param1 = (int)my\_addr; *// 使用 my\_addr 作为 param1 参数*

flag和param1就被正确传递并使用。

1. 释放互斥锁并返回结果
2. mutex\_unlock(&peep\_page\_mutex);
3. return result;

含义：在关键区域结束后释放互斥锁，并返回计算结果。

**四、编译内核并调用自己的系统调用**

1.编译内核

确保所有修改都已经保存。

（1）进入内核源代码目录：

cd /path/to/kernel/source

（2）配置内核：

make menuconfig

确保新的系统调用已经包含在配置中。

（3）编译内核和模块：

1. make -j$(nproc)
2. make modules\_install
3. make install

（4）重启系统并选择新编译的内核。

（5）如遇到内核空间不够，需要及时清理旧内核和不需要的日志文件而不是扩容。

2. 调用系统调用

（1）编写用户空间程序 fine.c：

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <errno.h>
4. #include "peep\_page.h"
5. int main() {
6. int param1 = 894; *// 以整型数表示的学号后3位*
7. int flag = 0;
8. int result0 = peep\_page(param1, flag);
9. if (result0 == -1) {
10. perror("sys\_peep\_page with flag 0 failed");
11. } else {
12. printf("Result with flag 0: %d\n", result0); *// 应该输出学号后3位的个位*
13. }
14. flag = 1;
15. int result1 = peep\_page(param1, flag);
16. if (result1 == -1) {
17. perror("sys\_peep\_page with flag 1 failed");
18. } else {
19. printf("Result with flag 1: %d\n", result1); *// 应该输出学号后3位的十位*
20. }
21. return 0;
22. }

定义param1为894，表示学号后3位。

定义flag为0，表示将要获取param1的个位数。

调用peep\_page(param1, flag)，如果返回-1，打印错误信息；否则打印返回值（个位数）。

将flag设为1，表示将要获取param1的十位数。

再次调用peep\_page(param1, flag)，如果返回-1，打印错误信息；否则打印返回值（十位数）。

（2）编写头文件 peep\_page.h

1. #ifndef PEEP\_PAGE\_H
2. #define PEEP\_PAGE\_H
3. #include <unistd.h>
4. #include <sys/syscall.h>
5. #define \_\_NR\_peep\_page 548
6. static inline long peep\_page(int param1, int flag)
7. {
8. return syscall(\_\_NR\_peep\_page, param1, flag);
9. }
10. #endif

**-为什么需要peep\_page.h**

peep\_page.h是一个头文件，定义了用户空间如何调用我们自定义的系统调用。它的作用如下：

#define \_\_NR\_peep\_page 548

它定义了系统调用的编号548，确保用户空间程序知道调用哪个系统调用号。

内联函数peep\_page：

1. static inline long peep\_page(int param1, int flag)
2. {
3. return syscall(\_\_NR\_peep\_page, param1, flag);
4. }

该内联函数封装了syscall系统调用，方便用户空间程序调用自定义的系统调用，而不必直接使用syscall函数。这样可以提高代码的可读性和可维护性。

头文件保护：

1. #ifndef PEEP\_PAGE\_H
2. #define PEEP\_PAGE\_H
3. *// ...*
4. #endif

头文件保护防止头文件被重复包含，引起编译错误。

peep\_page.h头文件通过定义系统调用号和封装系统调用的内联函数，为用户空间程序提供了方便的接口，以便调用自定义的系统调用。

（3）编译运行用户空间程序

1. gcc -g -o fine fine.c
2. ./fine

输出结果如图所示：