添加系统调用

实验目的

学习重建 Linux 内核。

学习 Linux 内核的系统调用，理解、掌握 Linux 系统调用的实现框架、用户界面、参数传递、进入/返回过程。阅读 Linux 内核源代码，通过添加一个简单的系统调用实验，进一步理解 Linux 操作系统处理系统调用的统一流程。了解 Linux 操作系统缺页处理，进一步掌握 task_struct 结构的作用。

实验内容

在现有的系统中添加一个不用传递参数的系统调用。这个系统调用的功能是实现统计操作系统缺页总次数，当前进程的缺页次数。严格来说这里讲的“缺页次数”实际上是页错误次数，即调用 do_page_fault 函数的次数。实验主要内容：

- 在 Linux 操作系统环境下重建内核
- 添加系统调用的名字
- 利用标准 C 库进行包装
- 添加系统调用号
- 在系统调用表中添加相应表项
- 修改统计缺页次数
- sys_mysyscall 的实现
- 编写用户态测试程序

实验指导

说明:本实验指导基于老版本的 Ubuntu，实验环境、版本、命令仅作参考。本年度实验提交内容，要求使用新版本 Ubuntu 不低于 18.04（可选择 18.04，18.10，19.04），内核版本不低于 4.15，4.15 之后版本均可。

下面的指导是以 ubuntu 16.04 和 kernel 4.13.0 为例，不同的 Linux 发行版本和内核版本，实验方法可能会有所不同。添加新的系统调用的步骤：

如何重新编译内核见“实验指导 1__Linux 内核重建”

1. 在系统调用表中添加或修改相应表项

我们前面讲过，系统调用处理程序（system_call）会根据 eax 中的索引到系统调用表（sys_call_table）中去寻找相应的表项。所以，我们必须在那里添加我们自己的一个值。

```
330     common  pkey_alloc              sys_pkey_alloc
331     common  pkey_free               sys_pkey_free
332     common  statx                   sys_statx
333     common  mysyscall               sys_mysyscall
```

arch/x86/entry/syscalls/syscall_64.tbl

到现在为止，系统已经能够正确地找到并且调用 sys_mysyscall。剩下的就只有一件事情，那就是 sys_mysyscall 的实现。

2. 修改统计系统缺页次数和进程缺页次数的内核代码

由于每发生一次缺页都要进入缺页中断服务函数 do_page_fault 一次，所以可以认为执行该函数的次数就是系统发生缺页的次数。可以定义一个全局变量 pfcount 作为计数变量，在执行 do_page_fault 时，该变量值加 1。在当前进程控制块中定义一个变量 pf 记录当前进程缺页次数，在执行 do_page_fault 时，这个变量值加 1。

先在 include/linux/mm.h 文件中声明变量 pfcount：

++ extern unsigned long pfcount;

要记录进程产生的缺页次数，首先在进程 task_struct 中增加成员 pf，在 include/linux/sched.h 文件中的 task_struct 结构中添加 pf 字段：

```
        atomic_t                        tick_dep_mask;
endif
        /* Context switch counts: */
        unsigned long                   nvcsw;
        unsigned long                   nivcsw;

        unsigned  long  pf;
```

统计当前进程缺页次数需要在创建进程是需要将进程控制块中的 pf 设置为 0，在进程创建过程中，子进程会把父进程的进程控制块复制一份，实现该复制过程的函数是 kernel/fork.c 文件中的 dup_task_struct()函数，修改该函数将子进程的 pf 设置成 0：

```
static struct task_struct *dup_task_struct(struct task_struct *orig, int node)
{
        struct task_struct *tsk;
        unsigned long *stack;
        struct vm_struct *stack_vm_area;
        int err;

        if (node == NUMA_NO_NODE)
                node = tsk_fork_get_node(orig);
        tsk = alloc_task_struct_node(node);
        if (!tsk)
                return NULL;

        stack = alloc_thread_stack_node(tsk, node);
        if (!stack)
                goto free_tsk;

        tsk->pf = 0;

        stack_vm_area = task_stack_vm_area(tsk);

        err = arch_dup_task_struct(tsk, orig);
```

在 arch/x86/mm/fault.c 文件中定义变量 pfcount；并修改 arch/x86/mm/fault.c 中 do_page_fault()函数。每次产生缺页中断，do_page_fault()函数会被调用，pfcount 变量值递增 1,记录系统产生缺页次数，current->pf 值递增 1，记录当前进程产生缺页次数：

```
 */
unsigned long pfcount;
static noinline void
__do_page_fault(struct pt_regs *regs, unsigned long error_code,
                unsigned long address)
{
        struct vm_area_struct *vma;
        struct task_struct *tsk;
        struct mm_struct *mm;
        int fault, major = 0;
        unsigned int flags = FAULT_FLAG_ALLOW_RETRY | FAULT_FLAG_KILLABLE;
        u32 pkey;

        tsk = current;
        mm = tsk->mm;

        pfcount++;
        current->pf++;

        /*
```

3. sys_mysyscall 的实现

我们把这一小段程序添加在 kernel/sys.c 里面。在这里，我们没有在 kernel 目录下另外添加自己的一个文件，这样做的目的是为了简单，而且不用修改 Makefile，省去不必要的麻烦。

mysyscall 系统调用实现输出。比如如下代码打印出当前进程缺页的次数。

```
/* Thread ID - the internal kernel "pid" */
SYSCALL_DEFINE0(gettid)
{
        return task_pid_vnr(current);
}

SYSCALL_DEFINE0(mysyscall)
{
        printk("current process - page fault count %ld \n", current->pf);
        return 0;
}
```

4. 编译内核和重启内核

重新编译内核，参加上一次实验指导。

make -j4
sudo make modules_install -j 4
sudo make install -j 4

如果编译过程中产生错误，你需要检查修改的代码是否正确，修改后再次编译，直至编译成功。

我们编译安装好了内核和模块后，生成可以 boot 的 initrd.img

sudo update-initramfs -c -k 4.13.16

最后更新启动文件

#sudo update-grub

我们已经编译了内核放到了指定位置/boot。现在，请你重启主机系统，期待编译过的 Linux 操作系统内核正常运行！

sudo reboot

5. 编写用户态程序

要测试新添加的系统调用，需要编写一个用户态测试程序（test.c）调用 mysyscall 系统调用。mysyscall 系统调用中 printk 函数输出的信息在/var/log/messages 文件中(ubuntu 为/var/log/kern.log 文件)。/var/log/messages (ubuntu 为/var/log/kern.log 文件)文件中的内容也可以在 shell 下用 dmesg 命令查看到。

```
#include <linux/unistd.h>
#include <sys/syscall.h>
#define __NR_mysyscall  333

int main()
{
syscall(__NR_mysyscall);
}
```

- 用 gcc 编译源程序

  gcc –o test test.c
- 运行程序

  ./test

```
[  110.128194] current process - page fault count 10320
[  117.586904] current process - page fault count 10540
```

完成实验后回答问题，并上交实验报告：

说明:本实验指导基于老版本的 Ubuntu，实验环境、版本、命令仅作参考。本年度实验提交内容，要求使用新版本 Ubuntu 不低于 18.04（可选择 18.04，18.10，19.04），内核版本不低于 4.15，4.15 之后版本均可。

1. 在 test.c 中添加打印整个系统 page fault 的变量值（也就是 pfcount 的值）。上传你的 test.c 代码。
2. 运行 test 程序后，dmesg 的截图证明你的系统调用添加成功，并且能在用户态被调用。
3. 多次运行 test 程序，每次运行 test 后记录下系统缺页次数和当前进程缺页次数，
4. 除了通过修改内核来添加一个系统调用外，还有其他的添加或修改一个系统调用的方法吗？如果有，请论述。
5. 对于一个操作系统而言，你认为修改系统调用的方法安全吗？请发表你的观点。
6. 在实验过程中遇到了什么问题，你是如何解决的。