添加系统调用

实验目的

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和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：

- 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
- 修改统计缺页次数相关的内核结构和函数
- sys_mysyscall 的实现
- 编写用户态测试程序

实验指导

本实验指导的实验环境与实验 2 相同。一个添加新的系统调用的步骤，：

1. 下载并部署内核源代码

   在实验 2 的基础上，不需要做这一步了。

2. 添加系统调用号

系统调用号在文件 unistd.h 里面定义。这个文件可能在你的 Linux 系统上会有**两个版本**：一个是 C 库文件版本，出现的地方是在 **ubuntu 13.04 只修改 /usr/include/asm-generic/unistd.h**；另外还有一个版本是内核自己的 unistd.h，出现的地方是在你解压出来的内核代码的对应位置（比如 **include/uapi/asm-generic/unistd.h**）。当然，也有可能这个 C 库文件只是一个到对应内核文件的连接。现在，你要做的就是在文件 unistd.h 中添加我们的系统调用号：__NR_mysyscall，x86 体系架构的系统调用号 223 没有使用，我们新的系统调用号定义为 223 号，如下所示：

kernel 3.11.4 为：include/uapi/asm-generic/unistd.h

ubuntu 13.04 为：/usr/include/asm-generic/unistd.h

在/usr/include/asm-generic/unistd.h 文件中的查找定义 223 号的行，作如下修改：

-- #define __NR3264_fadvise64 223
--__SC_COMP(__NR3264_fadvise64, sys_fadvise64_64, compat_sys_fadvise64_64)

++ #define __NR_mysyscall 274
++ __SYSCALL(__NR_mysyscall, sys_mysyscall)

在文件 include/uapi/asm-generic/unistd.h 中做同样的修改

注意：不同版本的内核，需要修改路径（子目录名和文件名）可能不一样，根据实际版本号查找这些.h 文件。系统调用号也可能不一样，可以根据内核版本不同对系统调用号进行修改

添加系统调用号之后，系统才能根据这个号，作为索引，去找 syscall_table 中的相应表项。所以说，我们接下来的一步就是：

3. 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。

arch/i386/kernel/syscall_table.S
(3.11.4 版本 kernel 修改 arch/x86/syscalls//syscall_32.tbl）

# 222 is unused		
223	**i386 mysyscall**	**sys_mysyscall**
224	**i386 gettid**	**sys_gettid**
225	**i386 readahead**	**sys_readahead**

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。

4. 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：

++ 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)
{
    …..
    tsk = alloc_task_struct_node(node);
    if (!tsk)
        return NULL;

    ++ tsk->pf=0;
    ……
}
```

在 arch/x86/mm/fault.c 文件中定义变量 pfcount，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 void __kprobes __do_page_fault(struct pt_regs *regs, unsigned long error_code)
{
        …
 ++ pfcount++;
 ++ current->pf++;
    …
}
```

5. 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 系统调用实现输出系统缺页次数和进程缺页次数。

```
asmlinkage int sys_mysyscall(void)

{
    ……
```

```
        return 0;
}
```

6. 重新编译内核

一定要重新编译内核。内核编译完成后，重新启动编译后的新内核。

7. 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。

用户态程序

```
#include <linux/unistd.h>
# include <sys/syscall.h>
#define __NR_ mysyscall    223
int main()
{
        syscall(__NR_mysyscall);       /*或 syscall(223)   */
          ……
}
```

- 用 gcc 编译源程序

  # **gcc –o test test.c**

- 运行程序

  # **./test**

回答问题：

1. 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，给出这些数据。test 程序打印的缺页次数是否就是操作系统原理上的缺页次数？有什么区别？
2. 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。
3. 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。

撰写实验报告的要求

1. 按照实验报告模板格式撰写；
2. **整个实验过程的截图；**
3. 源程序的修改部分，运行结果的截图；

4. 实验过程中遇到的问题及解决方法等；
5. 心得体会。