操作系统 作业2

- 王华强
- 2016K8009929035

作业指导

Linux 下常见的3种系统调用方法包括有：

1. 通过glibc提供的库函数
2. 使用syscall函数直接调用相应的系统调用
3. 通过 int 80指令陷入（32bit）或者通过syscall指令陷入（64bit）

请研究Linux(kernel>=2.6.24) gettimeofday这一系统调用的用法，并且选择上述3种系统调用方法中的2种来执行，记录其运行时间。

提示：请思考一次系统调用的时间开销的量级，对比结果，并尝试解释其中原因。

提交内容： 程序、执行结果、结果分析、系统环境（uname -a），本人学号。

实验背景: Linux系统调用

ref: https://www.cnblogs.com/yiyide266/p/5538079.html

它在<内核源码目录>/kernel/entry.S，Entry(system_call)的下一行。

实验环境构建

涉及到嵌入式汇编的使用, 用cpp构建实验程序. 使用 `clock()` 函数来完成计时. 将每种调用方式循环运行100000次, 经测试循环本身的开销要远远低于系统中断的数量级, 因此忽略不计. 三种系统调用的实现如下:

直接使用汇编描述系统调用

在系统中找到 `gettimeofday` 的位置如下:

```
cat `locate unistd_64` > log.log
```

结果是:

```
eval 'sub __NR_gettimeofday () {96;}' unless defined(&__NR_gettimeofday);
```

编写嵌入式汇编如下:

```
void asm_gettime2()
{
    __asm__(
        "push %rax\n\t"
        "push %rdi\n\t"
        "push %rsi\n\t"
        "mov $96,%rax\n\t"
        "mov tv,%rdi\n\t"
        "mov tz,%rsi\n\t"
        "syscall \n\t"
        "pop %rsi\n\t"
        "pop %rdi\n\t"
        "pop %rax\n\t"
    );
}
```

使用syscall函数直接调用相应的系统调用

```
syscall(SYS_gettimeofday, &tvs, 0);
```

系统调用定义可以在这里找到: `<sys/syscall.h>`

使用glibc提供的库函数

```
#include <sys/time.h>
gettimeofday(tv, 0);
```

测试结果

在不同平台上的测试结果如下:

Ubuntu 16.04 虚拟机

Linux cod-VirtualBox 4.15.0-25-generic #27~16.04.1-Ubuntu SMP Mon Jul 2 17:05:41 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux

测试的结果是:

```
time_start:1292
time_end:830910
time_macro:369563
time_glibc:38031
time_asm:422022
```

Windows Subsystem of Linux

Linux AW-OMEN 4.4.0-17713-Microsoft #1000-Microsoft Fri Jul 06 15:51:00 PST 2018 x86_64 x86_64 x86_64 GNU/Linux

```
time_start:15625
time_end:2328125
time_macro:1171875
time_glibc:1140625
time_asm:0//无法编译通过
```

结果分析

在linux平台上, 三种调用方式所用的时间在不同的数量级.

简单的阅读了一下manpage, 并没有发现什么有用的信息. 分别尝试反汇编三种实现方式, 比较其实现区别.

syscall做法的关键汇编指令如下:

```
400564:       e8 a7 fe ff ff          callq  400410 <syscall@plt>

0000000000400410 <syscall@plt>:
400410:       ff 25 0a 0c 20 00       jmpq   *0x200c0a(%rip)        # 601020 <_GLOBAL_OFFSET_TABLE_+0x20>
400416:       68 01 00 00 00          pushq  $0x1
40041b:       e9 d0 ff ff ff          jmpq   4003f0 <_init+0x28>

00000000004003f0 <__libc_start_main@plt-0x10>:
4003f0:       ff 35 12 0c 20 00       pushq  0x200c12(%rip)        # 601008 <_GLOBAL_OFFSET_TABLE_+0x8>
4003f6:       ff 25 14 0c 20 00       jmpq   *0x200c14(%rip)        # 601010 <_GLOBAL_OFFSET_TABLE_+0x10>
4003fc:       0f 1f 40 00             nopl   0x0(%rax)
#然后就不知道跳到哪里去了
```

glibc的汇编结果类似, 最后都跳转到全局偏移表中的某个位置, 追起来太麻烦了...

使用嵌入式汇编的汇编指令仅仅是引入了一些`nop`指令, 猜测仅仅是为了对齐流水, 在此没有必要展开.

可见分析汇编指令得到的信息有限, 可能应该尝试从原理上进行阐述.

理论上, 直接使用汇编指令所需要的时间最少. `syscall()`是原来的`__syscall`宏的替代品, 两者可能只起到简单的替换作用, 因此测试出来的结果大致相同. glibc封装系统调用并提供标准的库函数, 理论上抽象程度最高但是性能最低, 一个可能的原因是在使用glibc中的函数时经过了多层的函数调用. 不过这些分析没有办法完全解释为何三者的运行时间会有如此大的差距, 以及为什么glibc的运行时间出人意料地低.

一个解释是: glibc的实现根本没有使用系统调用, 不然不可能达到这样的速度.

> 总之我试着看了汇编, 看了源码(没看懂), 但是什么都没看出来. 我自闭了.....

在WSL平台上, 可用的两种调用方式有相同的时间数量级, 消耗时间基本相同, 且远远高于Linux下直接运行的耗时.

没有时间去阅读WSL的内核实现. 猜测是因为WSL的内核为了在Windows环境下工作进行了专门的改动, 且两种方式的实现原理相同.

附件

完整测试代码:

```c
#include <bits/stdc++.h>
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

struct timeval tvs;
struct timeval *tv = &tvs;
struct timezone tzs;
struct timezone *tz = &tzs;

// static __inline__ void asm_gettime(struct timeval *tv)
// {
//     __asm__(
//         "mov %%rax,96\n\t"
//         "mov %%rdi,%0\n\t"
//         "mov %%rsi,0\n\t"
//         "syscall \n\t"
//         : "=m"(tv)
//     );
// }

void asm_gettime2()
{
    __asm__(
        "push %rax\n\t"
        "push %rdi\n\t"
        "push %rsi\n\t"
        "mov $96,%rax\n\t"
        "mov tv,%rdi\n\t"
        "mov tz,%rsi\n\t"
        "syscall \n\t"
        "pop %rsi\n\t"
        "pop %rdi\n\t"
        "pop %rax\n\t");
}

// __asm__("\
    //     mov %rax,96\n\t\
    //     mov %rdi,0\n\t\
    //     mov %rsi,0\n\t\
    //     syscall \n\t\
    // ");

int main()
{
    long int time_start = clock();
    for (int i = 1000000; i-- > 0;)
        syscall(SYS_gettimeofday, &tvs, 0);
    long int time1 = clock();
    for (int i = 1000000; i-- > 0;)
        gettimeofday(tv, 0);
    long int time2 = clock();
    for (int i = 1000000; i-- > 0;)
        asm_gettime2();
    // asm_gettime(tv);
    long int time3 = clock();

    //SYS_gettimeofday(tv, tz);
    long int time_end = clock();

    printf("time_start:%ld\n", time_start);
    printf("time_end:%ld\n", time_end);
    printf("time_macro:%ld\n", time1 - time_start);
```

```
    printf("time_glibc:%ld\n", time2 - time1);
    printf("time_asm:%ld\n", time3 - time2);

    return 0;
}
```

补充:

系统调用的添加:

https://blog.csdn.net/xiaofanzidafanzi/article/details/52904406