OS_Assignment2

Assign	
Property	
tag	homework
姓名	周鹏宇
学号	2019K8009929039

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。

- 思路：进行多次系统调用，使用time命令记录时间
- 系统环境：

```
stu@stu:~/OShomo/OShomo2$ uname -a
Linux stu 5.4.0-81-generic #91-Ubuntu SMP Thu Jul 15 19:09:17 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux
```

1. 通过`glibc`提供的库函数

- 源代码

```
stu@stu:~/OShomo/OShomo2$ cat work1.c
#include <sys/syscall.h>
#include <sys/time.h>
#include <stdio.h>

int main() {
    struct timeval n;
    for (int i = 0; i < 11451400; i++) {
        gettimeofday(&n, NULL);
```

```
    }
    return 0;
}
```

- 运行结果

```
stu@stu:~/OShomo/OShomo2$ time ./work1

real  0m0.173s
user  0m0.172s
sys 0m0.000s
```

2. 使用 `syscall` 函数直接调用相应的系统调用

- 源代码

```
stu@stu:~/OShomo/OShomo2$ cat work2.c
#include <sys/syscall.h>
#include <sys/time.h>
#include <stdio.h>

int main() {
    struct timeval n;
    for (int i = 0; i < 11451400; i++) {
        syscall(SYS_gettimeofday, &n, NULL);
    }
    return 0;
}
```

- 运行结果

```
stu@stu:~/OShomo/OShomo2$ time ./work2

real  0m4.145s
user  0m3.135s
sys 0m1.009s
```

3. 通过 `int 80` 指令陷入（`32bit`）或者通过 `syscall` 指令陷入（`64bit`）

- 本来想通过32位汇编完成，但在`gcc -m32`后，不少库无法使用，只得使用64位
- 源代码

```
stu@stu:~/OShomo/OShomo2$ cat work3.c
#include <sys/syscall.h>
#include <sys/time.h>
```

```
#include <stdio.h>

int main() {
    struct timeval n;
    for (int i = 0; i < 11451400; i++) {
        asm volatile (
            "movq   %[syscall_id], %%rax \n"
            "movq   %[tv_addr], %%rdi \n"
            "movq   %[tz_addr], %%rsi \n"
            "int $0x80 \n"
            :
            : [syscall_id] "i" (SYS_gettimeofday),
              [tv_addr] "r" (&n),
              [tz_addr] "r" (NULL)
            : "rax", "rdi", "rsi"
        );
    }
    return 0;
}
```

- 运行结果

```
stu@stu:~/OShomo/OShomo2$ time ./work3

real  0m5.222s
user  0m3.171s
sys 0m2.050s
```

▼ 结论

1. 第一种系统调用的方法几乎没有内核态时间，时间总花销也远小于后二者
2. 第二种和第三种方法花销的时间相近，且内核态花费时间与用户态花费时间的比例相近，可能`system`函数的工作原理中包含了用`int 0x80`或者`syscall`实现软中断
3. 库函数调用无需陷入内核，其在用户地址空间执行，属于过程效用，开销较少；而后两者都涉及到陷入内核，在内核地址空间运行需要在用户空间和内核上下文环境间切换，开销较大。