操作系统-作业1

宋婉婷 2022K8009929009

1、任务说明

在 Linux 环境下，分别使用 glibc 、syscall 和 x86_64 内联汇编三种方法实现 getpid 和 open 系统调用，统计所用时间，并分析其中差异原因。

2、代码展示

实验环境为：

```
# uname -a
Linux 8a73e8e63827 5.15.167.4-microsoft-standard-WSL2 #1 SMP Tue Nov 5 00:21:55
UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
```

对于`gettimeofday`和`clock_gettime`，经过调研发现在精度上前者为微秒级，后者为纳秒级，而完成系统调用的时间较短，而且`clock_gettime`通过`VDSO`实现，性能更好，因此选择使用`clock_gettime`进行时间测量。

代码展示：

```
#include <stdio.h>
#include <time.h>
#include <unistd.h> //glibc
#include <sys/syscall.h> //syscall
#include <fcntl.h>

int main() {
    struct timespec ts1;
    struct timespec ts2;
    struct timespec ts3;
    struct timespec ts4;
    long pid;
    clock_gettime(CLOCK_MONOTONIC, &ts1);
    pid_t pid1 = getpid();
    clock_gettime(CLOCK_MONOTONIC, &ts2);
    pid_t pid2 = syscall(SYS_getpid);
    clock_gettime(CLOCK_MONOTONIC, &ts3);
    asm volatile (
        "mov $39, %%rax\n\t"
        "syscall\n\t"
        : "=a"(pid)
        :: "rcx", "r11"
    );
    clock_gettime(CLOCK_MONOTONIC, &ts4);

    printf("=========getpid==========\n");
    printf("glibc time is : %ld ns\n", ts2.tv_nsec - ts1.tv_nsec);
    printf("syscall time is : %ld ns\n", ts3.tv_nsec - ts2.tv_nsec);
    printf("asm time is : %ld ns\n", ts4.tv_nsec - ts3.tv_nsec);
```

```
    long fd;
    const char *path = "test.txt";

    clock_gettime(CLOCK_MONOTONIC, &ts1);
    int fd1 = open("test.txt", O_RDONLY);
    clock_gettime(CLOCK_MONOTONIC, &ts2);
    int fd2 = syscall(SYS_open, "test.txt", O_RDONLY);
    clock_gettime(CLOCK_MONOTONIC, &ts3);
    asm volatile (
        "mov $2, %%rax\n\t"      // SYS_open = 2
        "mov %1, %%rdi\n\t"      // 文件路径（第1个参数）
        "mov $0, %%rsi\n\t"      // O_RDONLY = 0（第2个参数）
        "syscall\n\t"            // 触发系统调用
        "mov %%rax, %0\n\t"      // 返回值存入 fd
        : "=r"(fd)               // 输出
        : "r"(path)              // 输入
        : "rax", "rdi", "rsi", "rcx", "r11" // 破坏的寄存器
    );
    clock_gettime(CLOCK_MONOTONIC, &ts4);

    printf("=========open==========\n");
    printf("glibc time is : %ld ns\n", ts2.tv_nsec - ts1.tv_nsec);
    printf("syscall time is : %ld ns\n", ts3.tv_nsec - ts2.tv_nsec);
    printf("asm time is : %ld ns\n", ts4.tv_nsec - ts3.tv_nsec);

    return 0;
}
```

3、结果分析

统计程序执行10次的结果，计算平均值如下：

		getpid			**open**	
	glibc	syscall	asm	glibc	syscall	asm
1	531	266	147	3273	1381	657
2	393	188	139	2154	773	825
3	1799	191	140	1927	726	1253
4	1699	191	141	1692	686	575
5	337	196	140	1544	689	639
6	1767	196	144	1704	703	594
7	361	209	148	1747	715	1193
8	366	180	128	1571	712	605
9	1864	187	143	1760	1155	885
10	1869	294	143	3043	703	1001
平均值	**1098.6**	**209.8**	**141.3**	**2041.5**	**824.3**	**822.7（ns）**

观察发现：

1. 对同一个系统调用，glibc 用时最长，syscall 和汇编时间都较短。在 getpid 调用上内联汇编时间最短，而 open 调用上两者平均差异不大。猜想：glibc 由于缓存机制的移除，引入了封装层处理参数、错误码转换等额外开销，因此所耗时间较长；而 syscall 和汇编绕过了封装层，因此时间开销较小。同时内联汇编相比于 syscall 时间更短，因为它可以一直在用户态运行，利用 vDSO（虚拟动态共享对象）的机制省去了上下文切换，更加灵活。
2. 对同样使用内联汇编调用，getpid 比 open 时间明显更短。猜想：getpid 的汇编代码实现简单，只需要读取进程描述符中的一个字段`pid`，不需要较多内存读写；而且 open 需要执行完整内核陷入流程，开销较大，破坏的寄存器也更多，导致时间较长。