几种backtrace方法

backtrace就是回溯程序的函数栈，我们在代码调试中经常会遇到，现总结下笔者所知道的以下四种backtrace方式：

1、直接调用libc函数 int backtrace(void**, int)

2、通过gcc内置函数__builtin_return_address获取函数返回地址，从而得到栈信息，只能是默认优化等级，即-O0，否则无法使用

3、直接内嵌汇编指令，读取寄存器数据，获取函数返回地址， 和2一样，只能-O0

4、使用libunwind.so接口函数int unw_backtrace (void**, int)

对比上述4种方式的性能， 1）的性能最差，和2）、3）、4）差一个数量级； 2）和3）性能最好， 性能是4）的3倍多

具体测试代码如下：

```
#include <execinfo.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <thread>
#include <dlfcn.h>

/*  gcc内置函数 */
/*********************************************/
#define GCCBT(i) \
    if(i >= size) return i;\
    frameNow = __builtin_frame_address(i);\
    if((unsigned long)frameNow <= (unsigned long)frameLast ||\
    ((unsigned long)frameLast != 0 && (unsigned long)frameNow > (unsigned long)frameLast + (1U
        return i;\
    frameLast = frameNow;\
    stack[i] = __builtin_extract_return_addr(__builtin_return_address(i));\

static int gccBacktrace(void** stack, int size)
{
    void* frameNow = 0;
    void* frameLast = 0;
    GCCBT(0); GCCBT(1); GCCBT(2); GCCBT(3); GCCBT(4); GCCBT(5); GCCBT(6); GCCBT(7);
    GCCBT(8); GCCBT(9); GCCBT(10); GCCBT(11); GCCBT(12); GCCBT(13); GCCBT(14); GCCBT(15);
    GCCBT(16); GCCBT(17); GCCBT(18); GCCBT(19); GCCBT(20); GCCBT(21); GCCBT(22); GCCBT(23);
    GCCBT(24); GCCBT(25); GCCBT(26); GCCBT(27); GCCBT(28); GCCBT(29); GCCBT(30); GCCBT(31);
    return 32;
}

/*  内置汇编获取寄存器返回值 */
/*********************************************/
```

```
static int asmBacktrace(void** stack, int size)
{
#ifndef __x86_64__
    return 0;
#else
    int frame = 0;
    void ** ebp;
    unsigned long long func_frame_distance = 0;
    __asm__ __volatile__("mov %%rbp, %0;\n\t" : "=m"(ebp) ::"memory");
    while (ebp && frame < size
        && (unsigned long long)(*ebp) < (unsigned long long)(ebp) + (1ULL << 24)//16M
        && (unsigned long long)(*ebp) > (unsigned long long)(ebp))
    {
        stack[frame++] = *(ebp + 1);
        ebp = (void**)(*ebp);
    }
    return frame;
#endif
}

static int flag = 1;
static int(*pfnBt)(void**, int);
void func5()
{
    void *btbuf[16];
    int btnum = pfnBt(btbuf, 16);
    if (flag == 1)
    {
        flag = 2;
        char **strings = backtrace_symbols(btbuf, btnum);
        for (int i = 0; i < btnum; ++i)
        {
            printf("%p %s\n", btbuf[i], strings[i]);
        }
        free(strings);
    }
}
void func4()
{
    func5();
}

void func3()
{
    func4();
}
void func2()
{
    func3();
}
void func1(int num)
{
    for (int i = 0; i < num; ++i)
    {
        func2();
```

```
    }
}

int main(int argc, char** argv)
{
    if (argc != 3)
    {
        printf("param1: int -- each thread loop number\n");
        printf("param2: string --bt method libc/gcc/asm/unw\n");
        return -1;
    }
    int loopNum = atoi(argv[1]);
    std::string method = argv[2];
    if (method == "libc")
    {
        printf("test libc\n");
        pfnBt = backtrace;
    }
    else if (method == "gcc")
    {
        printf("test gcc\n");
        pfnBt = gccBacktrace;
    }
    else if (method == "asm")
    {
        printf("test asm\n");
        pfnBt = asmBacktrace;
    }
    else if (method == "uwn")
    {
        printf("test uwn\n");
        void* handle = dlopen("./libunwind.so",RTLD_LAZY|RTLD_LOCAL);
        if (handle == NULL)
        {
            printf("dlopen failed, %s\n", dlerror());
            return -1;
        }
        pfnBt = (int(*)(void**, int))dlsym(handle, "unw_backtrace");
        if (pfnBt == NULL)
        {
            printf("dlsym failed, %s\n", dlerror());
            return -1;
        }
    }
    else
    {
        printf("please input right method lic/gcc/asm/uwn\n");
        return -1;
    }
    std::thread thd[20];
    for (int i = 0; i < 20; ++i)
    {
        thd[i] = std::thread(func1, loopNum);
    }
    for (int i = 0; i < 20; ++i)
```

```
    {
        thd[i].join();
    }
    return 0;
}
```

编译命令：

```
g++ bt.cpp -o bt -lpthread -ldl -std=c++11 -g
```

对比测试效果截图如下：

```
[root@192 backtrace]# time ./bt 10000 libc
test libc
0x40296f ./bt() [0x40296f]
0x402a01 ./bt() [0x402a01]
0x402a0c ./bt() [0x402a0c]
0x402a17 ./bt() [0x402a17]
0x402a32 ./bt() [0x402a32]
0x404326 ./bt() [0x404326]
0x404233 ./bt() [0x404233]
0x4041cc ./bt() [0x4041cc]
0x7f72c6991330 /lib64/libstdc++.so.6(+0xb5330) [0x7f72c6991330]
0x7f72c6defea5 /lib64/libpthread.so.0(+0x7ea5) [0x7f72c6defea5]
0x7f72c60f4b0d /lib64/libc.so.6(clone+0x6d) [0x7f72c60f4b0d]

real    0m0.630s
user    0m0.599s
sys     0m0.002s

[root@192 backtrace]# time ./bt 10000 uwn
test uwn
0x40296e ./bt() [0x40296e]
0x402a00 ./bt() [0x402a00]
0x402a0b ./bt() [0x402a0b]
0x402a16 ./bt() [0x402a16]
0x402a31 ./bt() [0x402a31]
0x404325 ./bt() [0x404325]
0x404232 ./bt() [0x404232]
0x4041cb ./bt() [0x4041cb]
0x7f97098c932f /lib64/libstdc++.so.6(+0xb532f) [0x7f97098c932f]
0x7f9709d27ea4 /lib64/libpthread.so.0(+0x7ea4) [0x7f9709d27ea4]
0x7f970902cb0c /lib64/libc.so.6(clone+0x6c) [0x7f970902cb0c]

real    0m0.055s
user    0m0.036s
sys     0m0.005s
```

```
[root@192 backtrace]# time ./bt 10000 asm
test asm
0x40296f ./bt() [0x40296f]
0x402a01 ./bt() [0x402a01]
0x402a0c ./bt() [0x402a0c]
0x402a17 ./bt() [0x402a17]
0x402a32 ./bt() [0x402a32]
0x404326 ./bt() [0x404326]
0x404233 ./bt() [0x404233]
0x4041cc ./bt() [0x4041cc]

real    0m0.017s
user    0m0.011s
sys     0m0.001s
```

```
[root@192 backtrace]# time ./bt 10000 gcc
test gcc
0x40296f ./bt() [0x40296f]
0x402a01 ./bt() [0x402a01]
0x402a0c ./bt() [0x402a0c]
0x402a17 ./bt() [0x402a17]
0x402a32 ./bt() [0x402a32]
0x404326 ./bt() [0x404326]
0x404233 ./bt() [0x404233]
0x4041cc ./bt() [0x4041cc]
0x7f38e3883330 /lib64/libstdc++.so.6(+0xb5330) [0x7f38e3883330]

real    0m0.018s
user    0m0.008s
sys     0m0.006s
```

分类: linux工具

ho966
粉丝 - 0 关注 - 0

« 上一篇: 申请内存有哪些函数
» 下一篇: kafka如何单机部署

posted @ 2023-02-01 22:31 ho966 阅读(293) 评论(0) 编辑 收藏 举报