

- 一、前言
- 二、Linux 平台
 - 1. 注册异常信号的处理函数
 - 2. 捕获异常，获取函数调用栈信息
- 三、Windows 平台
 - 1. 设置异常处理函数
 - 2. 捕获异常，获取函数调用栈信息

- 一、前言
- 二、Linux 平台
 - 1. 注册异常信号的处理函数
 - 2. 捕获异常，获取函数调用栈信息
- 三、Windows 平台
 - 1. 设置异常处理函数
 - 2. 捕获异常，获取函数调用栈信息

一、前言

程序在执行过程中 **crash** 是非常严重的问题，一般都应该在**测试阶段**排除掉这些问题，但是总会有漏网之鱼被带到 release 阶段。

因此，程序的**日志系统**需要侦测这种情况，在代码崩溃的时候获取**函数调用栈**信息，为 debug 提供有效的信息。

这篇文章的理论知识很少，直接分享 **2 段代码**：在 **Linux** 和 **Windows** 这 2 个平台上，如何用 **C++** 来捕获函数调用栈里的信息。

二、Linux 平台

1. 注册异常信号的处理函数

需要处理哪些异常信号

```
#include <execinfo.h>
#include <cxxabi.h>
#include <signal.h>

const std::map<int, std::string> Signals = {
    {SIGINT, "SIGINT"},
    {SIGABRT, "SIGABRT"},
    {SIGFPE, "SIGFPE"},
    {SIGILL, "SIGILL"},
    {SIGSEGV, "SIGSEGV"}
    // 可以添加其他信号
};
```

注册信号处理函数

```
struct sigaction action;
sigemptyset(&action.sa_mask);
action.sa_sigaction = &sigHandler;
action.sa_flags = SA_SIGINFO;

for (const auto &sigPair : Signals)
{
    if (sigaction(sigPair.first, &action, NULL) < 0)
        fprintf(stderr, "Error: sigaction failed! \n");
}
```

2. 捕获异常，获取函数调用栈信息

```
void sigHandler(int signum, siginfo_t *info, void *ctx)
{
    const size_t dump_size = 50;
    void *array[dump_size];
    int size = backtrace(array, dump_size);
    char **symbols = backtrace_symbols(array, size);
    std::ostringstream oss;

    for (int i = 0; i < size; ++i)
    {
        char *mangleName = 0;
        char *offsetBegin = 0;
        char *offsetEnd = 0;

        for (char *p = symbols[i]; *p; ++p)
        {
            if ('(' == *p)
            {
                mangleName = p;
            }
            else if ('+' == *p)
            {
                offsetBegin = p;
            }
            else if (')' == *p)
            {
                offsetEnd = p;
                break;
            }
        }

        if (mangleName && offsetBegin && offsetEnd && mangleName < offsetBegin)
        {
            *mangleName++ = '\\0';
            *offsetBegin++ = '\\0';
            *offsetEnd++ = '\\0';

            int status;
            char *realName = abi::__cxa_demangle(mangleName, 0, 0, &status);
            if (0 == status)
                oss << "\\tstack dump [" << i << "]" << " " << symbols[i] << " : " << realName <<
"+";

            else
                oss << "\\tstack dump [" << i << "]" << " " << symbols[i] << mangleName << "+";
            oss << offsetBegin << offsetEnd << std::endl;
            free(realName);
        }
        else
        {
            oss << "\\tstack dump [" << i << "]" << " " << symbols[i] << std::endl;
        }
    }
    free(symbols);
    oss << std::endl;
    std::cout << oss.str(); // 打印函数调用栈信息
}
```

三、Windows 平台

在 Windows 平台下的代码实现，参考了国外某个老兄的代码，如下：

1. 设置异常处理函数

```
#include <windows.h>
#include <dbghelp.h>

SetUnhandledExceptionFilter(exceptionHandler);
```

2. 捕获异常，获取函数调用栈信息

```
void exceptionHandler(LPEXCEPTION_POINTERS info)
{
    CONTEXT *context = info->ContextRecord;
    std::shared_ptr<void> RaiiSysCleaner(nullptr, [&](void *) {
        SymCleanup(GetCurrentProcess());
    });

    const size_t dumpSize = 64;
    std::vector<uint64_t> frameVector(dumpSize);

    DWORD machine_type = 0;
    STACKFRAME64 frame = {};
    frame.AddrPC.Mode = AddrModeFlat;
    frame.AddrFrame.Mode = AddrModeFlat;
    frame.AddrStack.Mode = AddrModeFlat;

#ifdef _M_IX86
    frame.AddrPC.Offset = context->Eip;
    frame.AddrFrame.Offset = context->Ebp;
    frame.AddrStack.Offset = context->Esp;
    machine_type = IMAGE_FILE_MACHINE_I386;
#elif _M_X64
    frame.AddrPC.Offset = context->Rip;
    frame.AddrFrame.Offset = context->Rbp;
    frame.AddrStack.Offset = context->Rsp;
    machine_type = IMAGE_FILE_MACHINE_AMD64;
#elif _M_IA64
    frame.AddrPC.Offset = context->StIIP;
    frame.AddrFrame.Offset = context->IntSp;
    frame.AddrStack.Offset = context->IntSp;
    machine_type = IMAGE_FILE_MACHINE_IA64;
    frame.AddrBStore.Offset = context->RSBSP;
    frame.AddrBStore.Mode = AddrModeFlat;
#else
    frame.AddrPC.Offset = context->Eip;
    frame.AddrFrame.Offset = context->Ebp;
    frame.AddrStack.Offset = context->Esp;
    machine_type = IMAGE_FILE_MACHINE_I386;
#endif

    for (size_t index = 0; index < frameVector.size(); ++index)
    {
        if (StackWalk64(machine_type,
            GetCurrentProcess(),
            GetCurrentThread(),
            &frame,
```

```

        context,
        NULL,
        SymFunctionTableAccess64,
        SymGetModuleBase64,
        NULL)) {
    frameVector[index] = frame.AddrPC.Offset;
} else {
    break;
}
}

std::string dump;
const size_t kSize = framevector.size();
for (size_t index = 0; index < kSize && framevector[index]; ++index) {
    dump += getSymbolInfo(index, framevector);
    dump += "\n";
}

std::cout << dump;
}

```

主要是利用了 [StackWalk64](#) 这个函数，从地址转换为函数名称。

好文章，[要转发](#)；越分享，越幸运！

[星标公众号](#)，能更快找到我！



添加“道哥”个人微信，
加入技术交流群。

公号：IOT物联网小镇，
关注 + 星标。

推荐阅读

- [1. C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻](#)
- [2. 原来gdb的底层调试原理这么简单](#)
- [3. 一步步分析-如何用C实现面向对象编程](#)
- [4. 都说软件架构要分层、分模块，具体应该怎么做\(一\)](#)
- [5. 都说软件架构要分层、分模块，具体应该怎么做\(二\)](#)

公众号【IoT物联网入门】