



Advanced Usage of the C++23 Stacktrace Library

James Pascoe

ADVANCED USAGE OF THE C++23 STACKTRACE LIBRARY

(STACKTRACES AND HOW TO USE THEM)

Jim (James) Pascoe

<http://www.james-pascoe.com>

james@james-pascoe.com

<http://jamespascoe.github.io/accu2024-stacktrace>

<https://github.com/jamespascoe/accu2024-example-code>

ACCU Bristol and Bath Meetup Coordinator

STACKTRACES, TRACEBACKS, CALL SEQUENCES

- Locate problems without a debugger
 - assertions, exceptions, unreachable code
- Introspection
 - object lifetimes e.g. RAI
 - useful with coroutines
- Signal tracing e.g. SEGV (but use with caution)

STACKTRACES IN C++

- C++23 gives us standardised stacktraces
 - [P0881R7: A Proposal to Add A Stacktrace Library](#)
- Complementary libraries:
 - [cpptrace](#): C++11, good treatment of signal-safety
 - [backward-cpp](#): stack trace pretty printing
 - [Boost::Stacktrace](#): mature, also some signal-safety

FIBONACCI (WITH STACKTRACE)

```
1 #include <iostream>
2 #include <format>
3 #include <stacktrace>
4
5 class fib_with_stacktrace {
6
7     std::stacktrace::size_type max_depth = 0;
8     std::stacktrace max_trace;
9
10 public:
11
12     fib_with_stacktrace() = default;
13
14     // Return the nth number in the fibonacci sequence
15     unsigned int fib(unsigned int n)
16     {
17         auto trace = std::stacktrace::current();
18         auto depth = trace.size();
19         if (trace.size() > max_depth) {
20             max_trace = trace;
21             max_depth = depth;
```

COMPILATION AND LINKING

- GCC build must be enabled with:
 - `--enable-libstdc++-backtrace=yes`
- 'libstdc++-backtrace' is the implementation:
 - `<gcc trunk>/libstdc++-v3/include/std/stacktrace`
 - `<gcc trunk>/libbacktrace`
- Compiler options:
 - `-g -lstdc++_libbacktrace`

FIBONACCI OUTPUT

```
1 0 1 1 2 3 5 8 13 21 34
2 Max stack depth was: 13
3 stacktrace.cpp:17 fib_with_stacktrace::fib(unsigned int)
4 stacktrace.cpp:24 fib_with_stacktrace::fib(unsigned int)
5 stacktrace.cpp:24 fib_with_stacktrace::fib(unsigned int)
6 stacktrace.cpp:24 fib_with_stacktrace::fib(unsigned int)
7 stacktrace.cpp:24 fib_with_stacktrace::fib(unsigned int)
8 stacktrace.cpp:24 fib_with_stacktrace::fib(unsigned int)
9 stacktrace.cpp:24 fib_with_stacktrace::fib(unsigned int)
10 stacktrace.cpp:24 fib_with_stacktrace::fib(unsigned int)
11 stacktrace.cpp:24 fib_with_stacktrace::fib(unsigned int)
12 stacktrace.cpp:38 main
13 :0 // __libc_start_main (<glibc trunk>/csu/libc-start.c:129)
14 :0 _start
15 :0
```

OBSERVATIONS

- Attach information to stack frames via hash support
 - `std::unordered_map<std::stacktrace_entry, T>`
- GDB is great for exploring
 - Tip: [Andrea Cardaci's GDB Dashboard](#)
- Observer for implementation defined native handle
 - `constexpr nh_type native_handle() const noexcept;`

```
accu2024-stacktrace — ssh 192.168.1.125 — 201x55
~/Library/Mobile Documents/com~apple~CloudDocs/Talks/accu2024-stacktrace — ssh 192.168.1.125

Output/messages
Assembly
~
~
~
~
0x00007ffff6fe7ba0  __libc_start_main+0 push %r13
0x00007ffff6fe7ba2  __libc_start_main+2 push %r12
0x00007ffff6fe7ba4  __libc_start_main+4 xor %eax,%eax
0x00007ffff6fe7ba6  __libc_start_main+6 push %rbp
0x00007ffff6fe7ba7  __libc_start_main+7 push %rbx
Breakpoints
[1] break at 0x0000000000403ae0 for _start hit 1 time
Expressions
History
Memory
Registers
    rax 0x0000000000000001      rbx 0x0000000000000000      rcx 0x0000000000424bd0      rdx 0x00007fffffe9b8      rsi 0x0000000000000001      rdi 0x0000000000403bc6      rbp 0x0000000000000000
    rsp 0x00007fffffe988      r8 0x0000000000424c40      r9 0x00007ffff7de3b40      r10 0x00000000000630010      r11 0x0000000000000000      r12 0x0000000000403ae0      r13 0x00007fffffe9b0
    r14 0x0000000000000000      r15 0x0000000000000000      rip 0x00007ffff6fe7ba0      eflags [ IF ]      r11 0x0000000000000000      r12 0x0000000000403ae0      r13 0x00007fffffe9b0
    es 0x00000000      fs 0x00000000      gs 0x00000000
Source
132     ElfW(auxv_t) *auxvec,
133 #endif
134     __typeof (main) init,
135     void (*fini) (void),
136     void (*rtld_fini) (void), void *stack_end)
137 {
138     /* Result of the 'main' function. */
139     int result;
140
141     __libc_multiple_libcs = &_dl_starting_up && !_dl_starting_up;
Stack
[0] from 0x00007ffff6fe7ba0 in __libc_start_main+0 at ../csu/libc-start.c:137
[1] from 0x0000000000403b0a in _start
Threads
[1] id 10557 name stacktrace-fib from 0x00007ffff6fe7ba0 in __libc_start_main+0 at ../csu/libc-start.c:137
Variables
arg main = 0x403bc6 <main(): {int (int, char **, char **)} 0x403bc6 <main(): argc = 1, argv = 0x7fffffe9b8: 47 '/', init = 0x424bd0 <__libc_csu_init>: {int (int, char **, char **)} 0x424bd0 <__libc_csu_init>, fini = 0x424c40 <__libc_csu_fini>: {void (void)} 0x424c40 <__libc_csu_fini>, rtld_fini = 0x7ffff7de3b40 <_dl_fini>: {void (void)} 0x7ffff7de3b40 <_dl_fini>, stack_end = 0x7fffffe9a8
loc result = <optimised out>, unwind_buf = {cancel_jmp_buf = {[0] = {jmp_buf = {[0] = 4220178, [1] = 140737337398393, [2] = 140737348935808, [3] = not_first_call = <optimised out>}}}
__libc_start_main (main=0x403bc6 <main(): argc=1, argv=0x7fffffe9b8, init=0x424bd0 <__libc_csu_init>, fini=0x424c40 <__libc_csu_fini>, rtld_fini=0x7ffff7de3b40 <_dl_fini>, stack_end=0x7fffffe9a8)
at ../csu/libc-start.c:137
warning: Source file is more recent than executable.
137
>>>
```

COROUTINE EXAMPLE

```
1 // This is based on the Boost.Beast HTTP coro server example.  
2 // See the following for documentation and details:  
3 //  
4 // https://www.boost.org/doc/libs/1_84_0/libs/beast/doc/html/  
5 // beast/examples.html#beast.examples.servers_advanced  
6  
7 #include <boost/asio/dispatch.hpp>  
8 #include <boost/asio/spawn.hpp>  
9 #include <boost/asio/strand.hpp>  
10 #include <boost/beast/core.hpp>  
11 #include <boost/beast/http.hpp>  
12 #include <boost/beast/version.hpp>  
13 #include <boost/config.hpp>  
14  
15 #include <iostream>  
16 #include <memory>  
17 #include <thread>  
18 #include <vector>  
19 #include <stacktrace>  
20 #include <format>  
21
```

COROUTINE OUTPUT

```
1 /home/pascoej/accu2024-example-code/coro_http_server.cpp:58
2     operator()
3 /home/pascoej/accu2024-example-code/coro_http_server.cpp:71
4     do_session()
5 /usr/local/include/c++/13.0.1/bits/invoke.h:61
6     void std::__invoke_implementation()
7 /usr/local/include/c++/13.0.1/bits/invoke.h:96
8     std::__invoke_result()
9 /usr/local/include/c++/13.0.1/functional:506
10    void std::_Bind()
11 /usr/local/include/c++/13.0.1/functional:591
12    void std::_Bind()
13 /usr/local/include/boost/asio/impl/spawn.hpp:1493
14     boost::asio::detail::old_spawn_entry_point()
15 /usr/local/include/boost/asio/impl/spawn.hpp:1485
16     boost::asio::detail::old_spawn_entry_point()
17 /usr/local/include/boost/asio/impl/spawn.hpp:158
18     boost::asio::detail::spawnedCoroutineThread::entry_point()
19 /usr/local/include/boost/coroutine/detail/pull_coroutine_object.hpp:2
20     boost::coroutines::detail::pull_coroutine_object()
21 /usr/local/include/boost/coroutine/detail/trampoline_pull.hpp:41
```

ASYNCHRONOUS SIGNAL SAFETY

- How do I get a stacktrace in a signal handler?
- Tricky to implement safely
 - Collect minimal information in the handler
 - Resolve outside (or in another process)
- `std::stacktrace` is not (currently) signal safe
 - But could be in the future
- See: `cpptrace` and Boost `safe_dump_to`

CONCLUSIONS

- Stacktraces are useful
 - Enrichment, introspection, object lifetimes
- Stack depth can be a useful proxy
 - Inexpensive to store - easy to chart
- Use with caution in signal handlers
- Be mindful of stacktrace output (extensive)
- Debugging starts in the design phase

RESOURCES

- Episode 336: C++ Weekly (Jason Turner)
- Sandor Dargo's Blog
- Boost::Stacktrace Documentation
- Signal safety:
 - Theoretical Async Signal Safety (Boost)
 - Signal-Safe Tracing (cpptrace)
 - signal-safety (Linux man page)