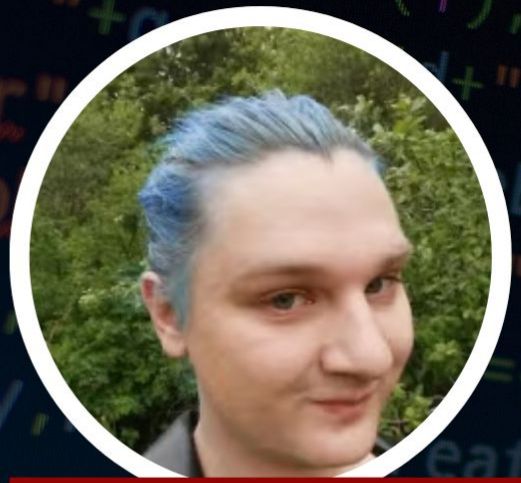
CityJS™

Segmentation fault! Low-level debugging for JS developers



Anna Henningsen

Who am I?

- Anna Henningsen
- she/her
- Staff Developer @ MongoDB
- Former full-time Node.js core contributor

@addaleax.bsky.social

https://addaleax.net/cityjs-london26.pdf

Earlier today, I learned…

node:ffi — Foreign Function Interface

24-ffi.mjs

```
// Node.js (proposed, nodejs/node#62072): node:ffi — foreign function interface.
// Call into shared libraries from pure JS — no node-gyp, no binding.gyp.
import { dlopen } from 'node:ffi';

// Resolve multiple symbols in one shot with typed signatures.
const { functions } = dlopen('./libmath.so', {
  add_i32:      { parameters: ['i32', 'i32'],    result: 'i32' },
  string_length: { parameters: ['pointer'],       result: 'u64' },
  hypot:        { parameters: ['f64', 'f64'],    result: 'f64' },
});

functions.add_i32(2, 3);        // => 5
functions.hypot(3, 4);          // => 5

// Behind `--experimental-ffi`. Gated by --allow-ffi in the Permission Model.
```

- Call into shared libraries from pure JS
- No node-gyp, no binding.gyp, no native addon rebuild
- Typed signatures: "i32", "f64", "pointer", "string", …
- --experimental-ffi, gated by --allow-ffi in the Permission Model

Stolen from Matteo's Slides!

nodejs/node#62072 · @cjihrig (Colin Ihrig) and @ShogunPanda (Paolo Insogna)

Earlier today, I learned...

- ... that this talk will be more relevant than I thought in the immediate future!

Earlier today, I learned...

- ... that this talk will be more relevant than I thought in the immediate future!
- ... that **I have job security for life**!

A few warnings

- This presentation will focus on Linux – it has the best DX, sorry
  - (some tools are cross-platform or at least have macOS equivalents)
- This presentation will focus on Node.js – it has the most users, sorry
  - (some approaches transfer to Deno/Bun)
- Tools may require system setup
  - Core dumps and debugging running processes may require editing system settings

Segmentation fault!

```
$ node test.js
Segmentation fault (core dumped)
```

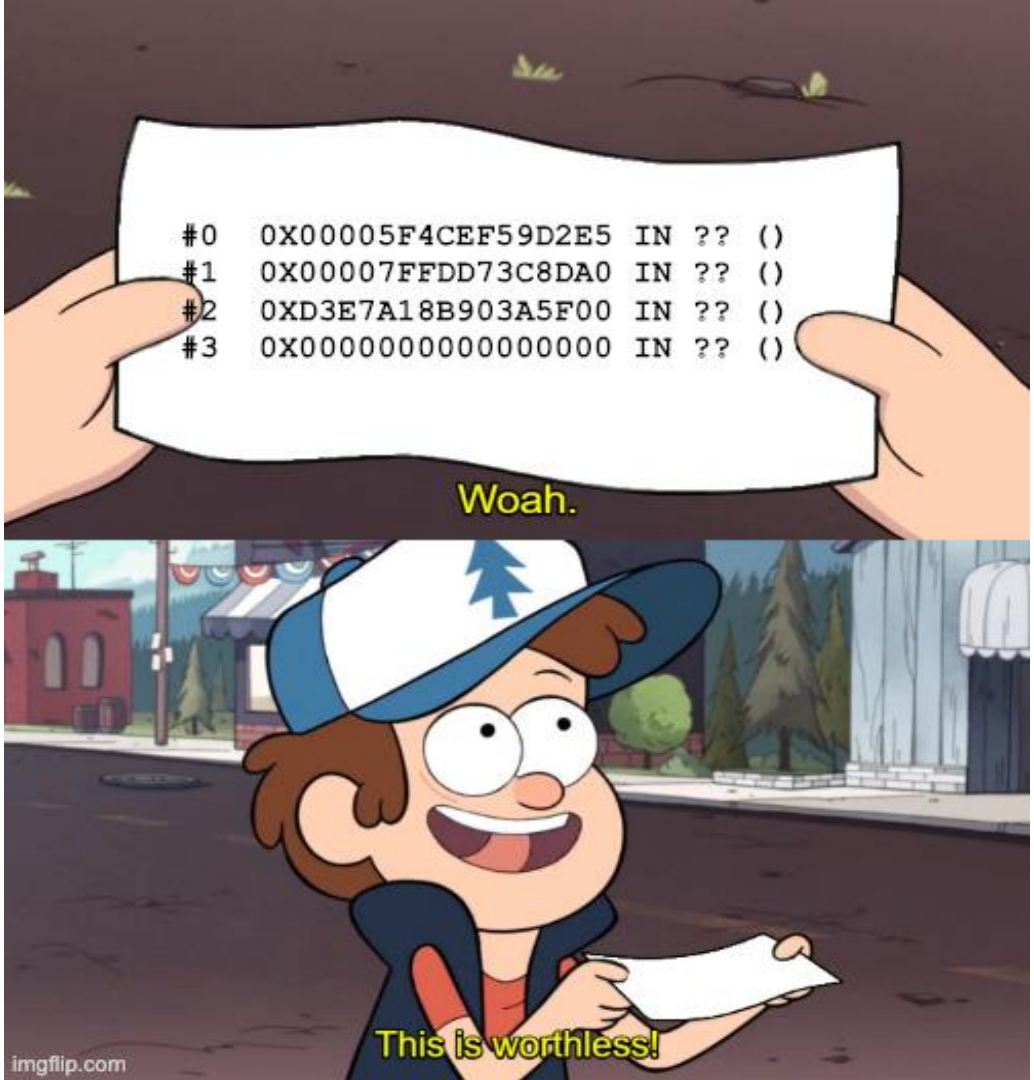
This is what C developers call an “error message”

What do I do with it?

```
$ node test.js
Segmentation fault (core dumped)
$ ls
core_node.101984  test.js # file name may vary!
```

Attempt 1: Using GDB on a core dump

```
$ gdb -c core_node.101984
[...]
Core was generated by `node test.js'.
Program terminated with signal SIGSEGV, Segmentation fault.
(gdb) backtrace
#0  0x00005f4cef59d2e5 in ?? ()
#1  0x00007ffdd73c8da0 in ?? ()
#2  0xd3e7a18b903a5f00 in ?? ()
#3  0x0000000000000000 in ?? ()
```


#0 0X00005F4CEF59D2E5 IN ?? ()
#1 0X00007FFDD73C8DA0 IN ?? ()
#2 0XD3E7A18B903A5F00 IN ?? ()
#3 0X0000000000000000 IN ?? ()
Woah.
This is worthless!
imgflip.com

Attempt 2: Include the executable!

```
$ gdb node -c core_node.101984
[...]
(gdb) backtrace
#0  0x00005f4cef59d2e5 in node::util::DefineLazyPropertiesGetter(v8::Local<v8::Name>,
v8::PropertyCallbackInfo<v8::Value> const&) ()
#1  0x00005f4cefc7285c in
v8::internal::Object::GetPropertyWithAccessor(v8::internal::LookupIterator*) ()
#2  0x00005f4cefc721c0 in
v8::internal::Object::GetProperty(v8::internal::LookupIterator*, bool) ()
#3  0x00005f4cefdd3c8e in
v8::internal::Runtime::GetObjectProperty(v8::internal::Isolate*,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>, bool*) ()
[...]
```

Attempt 2: Include the executable!

```
$ gdb node -c core_node.101984
[...]
(gdb) backtrace
#0  0x00005f4cef59d2e5 in node::util::DefineLazyPropertiesGetter(v8::Local<v8::Name>,
v8::PropertyCallbackInfo<v8::Value> const&) ()
#1  0x00005f4cefc7285c in
v8::internal::Object::GetPropertyWithAccessor(v8::internal::LookupIterator*) ()
#2  0x00005f4cefc721c0 in
v8::internal::Object::GetProperty(v8::internal::LookupIterator*, bool) ()
#3  0x00005f4cefdd3c8e in
v8::internal::Runtime::GetObjectProperty(v8::internal::Isolate*,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>, bool*) ()
[...]
```

This is a function I can look up!

Attempt 3: Get a debug build of Node.js …

```
$ git clone git@github.com:nodejs/node.git && cd node
$ ./configure --debug-node # or --debug for full debugging
$ make -j16
[...]
$ path/to/node test.js
Segmentation fault (core dumped)
```

Attempt 3: Get a debug build of Node.js ...

```
$ gdb node -c core_node.101984
[...]
(gdb) backtrace
#0  0x000060e13b1cff76 in node::Realm::isolate (this=0x0) at ../src/node_realm-inl.h:46
#1  0x000060e13b5ab5e7 in node::util::DefineLazyPropertiesGetter (name=..., info=...)
at ../src/node_util.cc:354
#2  0x000060e13bcf394c in
v8::internal::Object::GetPropertyWithAccessor(v8::internal::LookupIterator*) ()
#3  0x000060e13bcf32b0 in
v8::internal::Object::GetProperty(v8::internal::LookupIterator*, bool) ()
#4  0x000060e13be54d7e in
v8::internal::Runtime::GetObjectProperty(v8::internal::Isolate*,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>, bool*) ()
[...]
```

Attempt 3: Get a debug build of Node.js ...

```
$ gdb node -c core_node.101984
[...]
(gdb) backtrace
#0  0x000060e13b1cff76 in node::Realm::isolate (this=0x0) at ../src/node_realm-inl.h:46
#1  0x000060e13b5ab5e7 in node::util::DefineLazyPropertiesGetter (name=..., info=...)
at ../src/node_util.cc:354
#2  0x000060e13bcf394c in
v8::internal::Object::GetPropertyWithAccessor(v8::internal::LookupIterator*) ()
#3  0x000060e13bcf32b0 in
v8::internal::Object::GetProperty(v8::internal::LookupIterator*, bool) ()
#4  0x000060e13be54d7e in
v8::internal::Runtime::GetObjectProperty(v8::internal::Isolate*,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>,
v8::internal::Handle<v8::internal::Object>, bool*) ()
[...]
```

Proper function details and line numbers! ✨

Attempt 3: Get a debug build of Node.js …

```
(gdb) backtrace
#0  0x000060e13b1cff76 in node::Realm::isolate (this=0x0) at ../src/node_realm-inl.h:46
[...]
(gdb) frame 0
#0  0x000060e13b1cff76 in node::Realm::isolate (this=0x0) at
../src/node_realm-inl.h:46
46    return isolate_;
(gdb) frame 1
#1  0x000060e13b5ab5e7 in node::util::DefineLazyPropertiesGetter (name=..., info=...)
at ../src/node_util.cc:354
354   Isolate* isolate = realm->isolate();
(gdb) print realm
$1 = (node::Realm *) 0x0
```

Attempt 3: Get a debug build of Node.js …

```
(gdb) list
349   }
350
351   static void DefineLazyPropertiesGetter(
352         Local<v8::Name> name, const v8::PropertyCallbackInfo<Value>& info) {
353   Realm* realm = Realm::GetCurrent(info);
354   Isolate* isolate = realm->isolate();
355   auto context = isolate->GetCurrentContext();
356   Local<Value> arg = info.Data();
357   Local<Value> require_result;
358   if (!realm->builtin_module_require()
(gdb) p name
$2 = {<v8::LocalBase<v8::Name>> = {<v8::api_internal::IndirectHandleBase> = {location_
= 0x7ffcd01c0b80}, <No data fields>}, <v8::api_internal::StackAllocated<false>> = {<No
data fields>}, <No data fields>}
```

Attempt 3: Get a debug build of Node.js ...

```
(gdb) print name
$2 = {<v8::LocalBase<v8::Name>> = {<v8::api_internal::IndirectHandleBase> = {location_
= 0x7ffcd01c0b80}, <No data fields>}, <v8::api_internal::StackAllocated<false>> = {<No
data fields>}, <No data fields>}
```

Attempt 3: Get a debug build of Node.js …

```
(gdb) print name
$2 = {<v8::LocalBase<v8::Name>> = {<v8::api_internal::IndirectHandleBase> = {location_
= 0x7ffcd01c0b80}, <No data fields>}, <v8::api_internal::StackAllocated<false>> = {<No
data fields>}, <No data fields>}
```

Sure … what is this exactly?

Attempt 4: Run Node.js live + V8 gdbinit!

```
# get https://github.com/v8/v8/blob/main/tools/gdbinit and put it
# into ~/.gdbinit
$ gdb --args node test.js
(gdb) run
Thread 1 "node" received signal SIGSEGV, Segmentation fault.
0x00005bffb8bcff76 in node::Realm::isolate (this=0x0) at
../src/node_realm-inl.h:46
46  return isolate_;
(gdb) frame 1
(gdb) jlh name
0x1bf970496821: [String] in OldSpace: #TextEncoder
```

Attempt 5: llnode!

- https://github.com/nodejs/llnode
- Built on top of lldb rather than gdb
- Plugin specifically designed for Node.js applications

Attempt 5: llnode!

```
$ llnode node -c core_node.18151
[...]
(llnode) v8 bt
error: node 0x00876bbc: DW_TAG_member '_M_pod_data' refers to type 0x00000000008c13da
which extends beyond the bounds of 0x00876b66
 * thread #1: tid = 18151, 0x000060e0827cff76
node`node::Realm::isolate(this=0x0000000000000000) const at node_realm-inl.h:46:10,
name = 'node', stop reason = signal SIGSEGV
  * frame #0: 0x000060e0827cff76 node`node::Realm::isolate(this=0x0000000000000000)
const at node_realm-inl.h:46:10
     frame #1: 0x000060e082bab5e7
node`node::util::DefineLazyPropertiesGetter(name=Local<v8::Name> @ 0x00007ffca32bf018,
info=0x00007ffca32bf128) at node_util.cc:354:36
```

Attempt 5: llnode!

```
$ llnode node -c core_node.18151
[...]
(llnode) frame select 1
node`node::util::DefineLazyPropertiesGetter(name=Local<v8::Name> @ 0x00007ffca32bf018,
info=0x00007ffca32bf128) at node_util.cc:354:36
   351     static void DefineLazyPropertiesGetter(
   352           Local<v8::Name> name, const v8::PropertyCallbackInfo<Value>& info) {
   353     Realm* realm = Realm::GetCurrent(info);
-> 354     Isolate* isolate = realm->isolate();
   355     auto context = isolate->GetCurrentContext();
   356     Local<Value> arg = info.Data();
   357     Local<Value> require_result;
(llnode) v8 inspect *name.location_
0x11d30dd96821:<String: "TextEncoder">
```

Okay, but my code doesn’t crash that often

Okay, but my code doesn't crash that often

Let me introduce you to a very special Node.js flag:

`--abort-on-uncaught-exception`

Added in: v0.10.8

Aborting instead of exiting causes a core file to be generated for post-mortem analysis using a debugger (such as `lldb`, `gdb`, and `mdb`).

If this flag is passed, the behavior can still be set to not abort through `process.setUncaughtExceptionCaptureCallback()` (and through usage of the `node:domain` module that uses it).

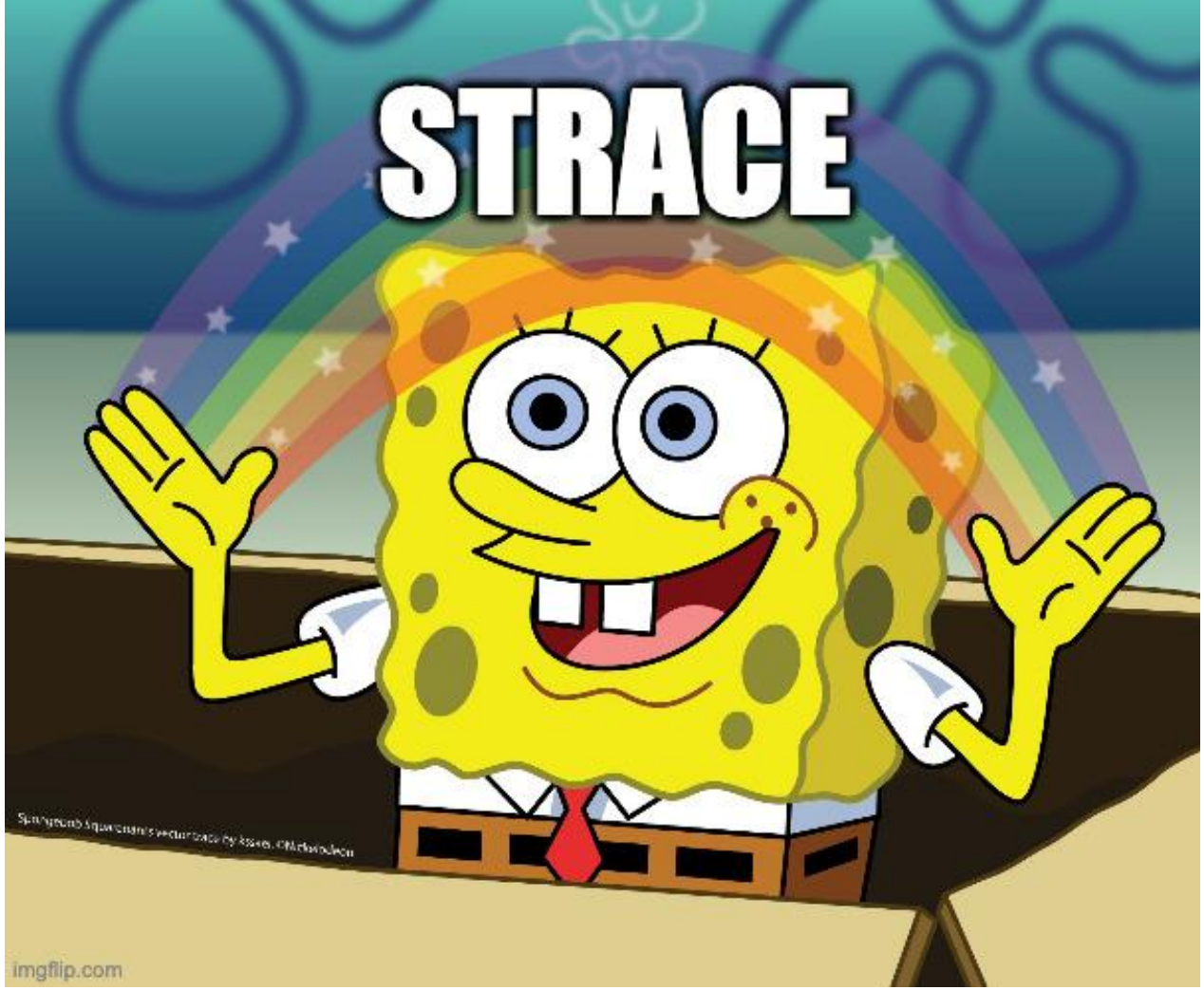
llnode can do more!

```
$ node --abort-on-uncaught-exception -e 'foo.bar()'
Uncaught ReferenceError: foo is not defined
[...]
$ llnode node -c core_MainThread.63251
(llnode) v8 bt
  * frame #0: 0x0000000001c044d2 node`v8::base::OS::Abort() + 18
[...]
     frame #5: 0x0000000000f33092
node`v8::internal::Runtime_LoadNoFeedbackIC_Miss(int, unsigned long*,
v8::internal::Isolate*) + 306
     frame #6: 0x00000000015d9e59 <exit>
     frame #7: 0x000000000165a7d3 <stub>
     frame #8: 0x000000000156c90a (this=0x35767fac1119:<Global proxy>) at [eval]:1:0
fn=0x000008864ad61d69
     frame #9: 0x000000000156ab18 <internal>
     frame #10: 0x000000000156a8a3 <entry>
```

My favorite debugging tool in the world

My favorite debugging tool in the world

- Can debug literally any application
- Observe every interaction of it with the outside world!
- Teaches you tons of things about how the OS works


STRACE
imgflip.com

Which files does my app access?

```
$ strace -e open,creat,openat,openat2 node -p '1+1'
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = openat(AT_FDCWD,
"/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/etc/ssl/openssl.cnf", O_RDONLY) = 3
openat(AT_FDCWD, "/proc/self/cgroup", O_RDONLY|O_CLOEXEC) = 17
openat(AT_FDCWD,
"/sys/fs/cgroup/user.slice/user-1000.slice/session-134.scope/memory.max",
O_RDONLY|O_CLOEXEC) = 17
[...]
```

Which hosts does my app connect to?

```
$ strace -f -e connect node -p 'fetch("http://example.com")'  2>&1 |
grep AF_INET
[pid 29569] connect(21, {sa_family=AF_INET, sin_port=htons( 53),
sin_addr=inet_addr(" 127.0.0.53")}, 16) = 0
[pid 29569] connect(21, {sa_family=AF_INET, sin_port=htons(0),
sin_addr=inet_addr("23.215.0.136")}, 16) = 0
[...]
[pid 29569] connect(21, {sa_family=AF_INET6, sin6_port=htons(0),
sin6_flowinfo=htonl(0), inet_pton(AF_INET6, "2600:1408:ec00:36::1736:7f24",
&sin6_addr), sin6_scope_id=0}, 28) = 0
[pid 29562] connect(21, {sa_family=AF_INET6, sin6_port=htons( 80),
sin6_flowinfo=htonl(0), inet_pton(AF_INET6, " 2600:1406:3a00:21::173e:2e66",
&sin6_addr), sin6_scope_id=0}, 28) = -1 EINPROGRESS (Operation now in progress)
```

What's in that HTTP request?

```
$ strace -f -s 1024 -e write,send,sendto node -p
'fetch("http://example.com")'  2>&1 | grep 'HTTP'
[pid 27573] write(18, "GET / HTTP/1.1\r\nhost:
example.com\r\nconnection: keep-alive\r\naccept:
*/*\r\naccept-language: *\r\nsec-fetch-mode: cors\r\nuser-agent:
node\r\naccept-encoding: gzip, deflate\r\n\r\n", 166) = 166
```

Are my file system accesses slow?

```
$ strace -f -ttt -e open,openat,close node -e
'fs.readdirSync("/usr", {recursive: true})'
[...]
[pid 50023] 1747928635.715108 openat(AT_FDCWD, "/usr/lib/llvm-14",
O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 17
[pid 50023] 1747928635.715272 close(17) = 0
[pid 50023] 1747928635.715524 openat(AT_FDCWD, "/usr/lib/llvm-18",
O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 17
[pid 50023] 1747928635.715688 close(17) = 0
[pid 50023] 1747928635.715945 openat(AT_FDCWD, "/usr/lib/locale",
O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 17
[pid 50023] 1747928635.716105 close(17) = 0
[...]
```

Let’s summarize!

- gdb and lldb are popular debuggers for native code

Let’s summarize!

- gdb and lldb are popular debuggers for native code
- Both have been extended to interact well with debugging mixed JS/C++ applications written for V8 (and by extension, Node.js/Deno)

Let's summarize!

- gdb and lldb are popular debuggers for native code
- Both have been extended to interact well with debugging mixed JS/C++ applications written for V8 (and by extension, Node.js/Deno)
- Both can be used to:
  - Debug core dumps from hard crashes (segmentation fault, abort, etc.)
  - Debug crashes due to uncaught exceptions with `--abort-on-uncaught-exception`
  - Debug running processes

Let’s summarize!

- gdb and lldb are popular debuggers for native code
- Both have been extended to interact well with debugging mixed JS/C++ applications written for V8 (and by extension, Node.js/Deno)
- Both can be used to:
  - Debug core dumps from hard crashes (segmentation fault, abort, etc.)
  - Debug crashes due to uncaught exceptions with `--abort-on-uncaught-exception`
  - Debug running processes
- Both benefit from having debug symbols available, either as part of the executable or externally

Let's summarize!

- gdb and lldb are popular debuggers for native code
- Both have been extended to interact well with debugging mixed JS/C++ applications written for V8 (and by extension, Node.js/Deno)
- Both can be used to:
  - Debug core dumps from hard crashes (segmentation fault, abort, etc.)
  - Debug crashes due to uncaught exceptions with `--abort-on-uncaught-exception`
  - Debug running processes
- Both benefit from having debug symbols available, either as part of the executable or externally
- `strace` is an incredibly useful tool in the right situations!

Thank you for your time.