Fuchsia

# Using Clang's source-based code coverage at scale

Gülfem Savrun Yeniçeri, Petr Hosek

gulfem@google.com phosek@google.com

# Code coverage in Fuchsia

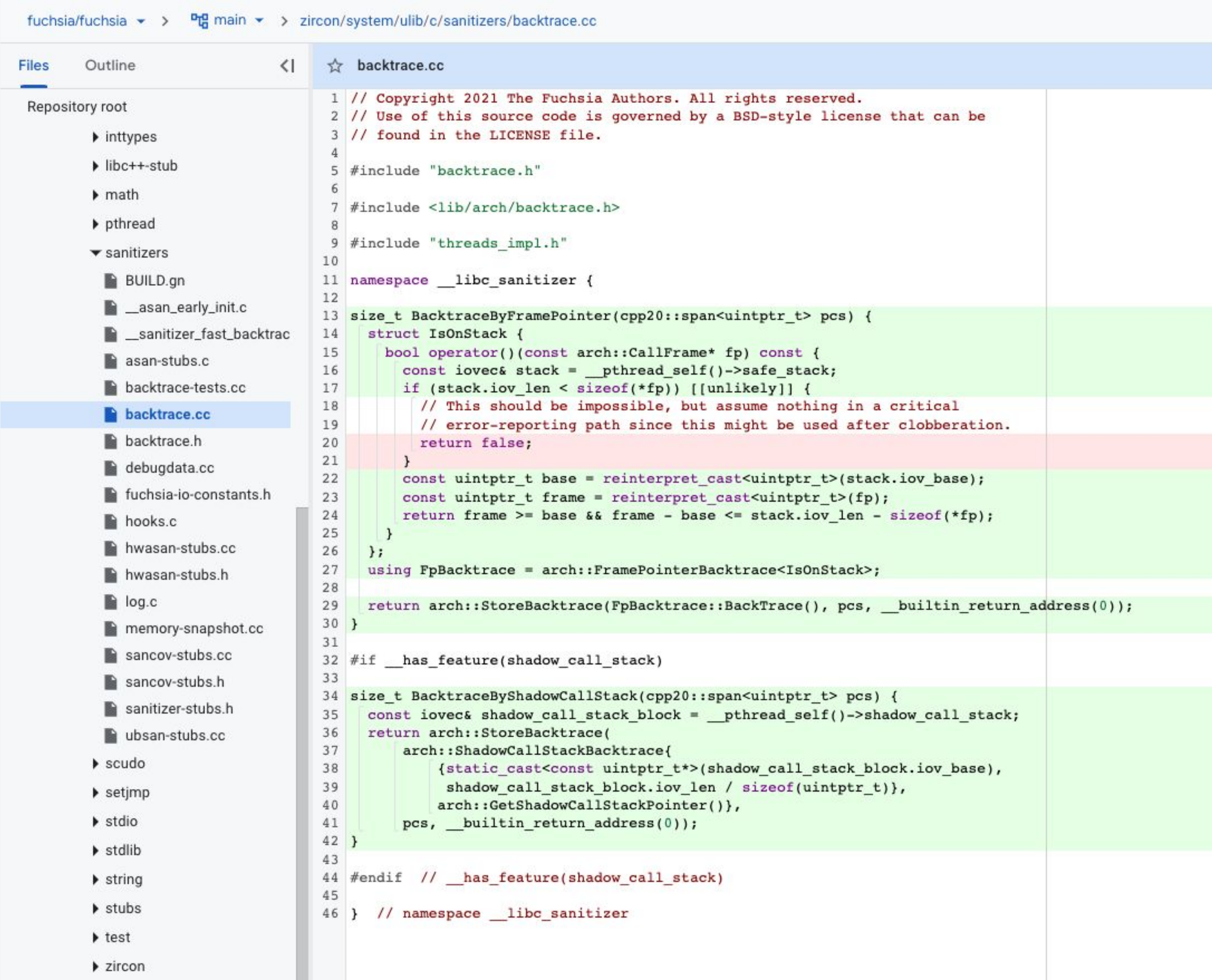
We collect incremental coverage at pre-submit testing and surface it in the code review tool.

zircon/system/ulib/c/sanitizers/backtrace.cc Added +90 -0

```
#include "backtrace.h"

#include "threads_impl.h"

namespace __libc_sanitizer {

size_t BacktraceByFramePointer(cpp20::span<uintptr_t> pcs) {
  struct FramePointer {
    const FramePointer* fp;
    uintptr_t pc;
  };

  auto on_stack = [&stack = __pthread_self()->safe_stack](const FramePointer* fp) -> bool {
    uintptr_t address = reinterpret_cast<uintptr_t>(fp);
    return address >= reinterpret_cast<uintptr_t>(stack.iov_base) &&
           address < reinterpret_cast<uintptr_t>(stack.iov_base) + stack.iov_len;
  };

  uintptr_t ra = reinterpret_cast<uintptr_t>(__builtin_return_address(0));
  auto fp = reinterpret_cast<const FramePointer*>(__builtin_frame_address(0));
  size_t i = 0;
  while (i < pcs.size() && on_stack(fp) && fp->pc != 0) {
    if (i == 0 && fp->pc != ra) {
      pcs[i++] = ra;
    } else {
      pcs[i++] = fp->pc;
      fp = fp->fp;
    }
  }
  if (i == 0 && i < pcs.size()) {
    pcs[i++] = ra;
  }

  return i;
}

#if __has_feature(shadow_call_stack)

namespace {

```

# Code coverage in Fuchsia

We collect absolute coverage in continuous integration and surface it in the code search tool.



# Clang source-based code coverage

Combines profiling (PGO) instrumentation with mapping derived from AST and preprocessor information.

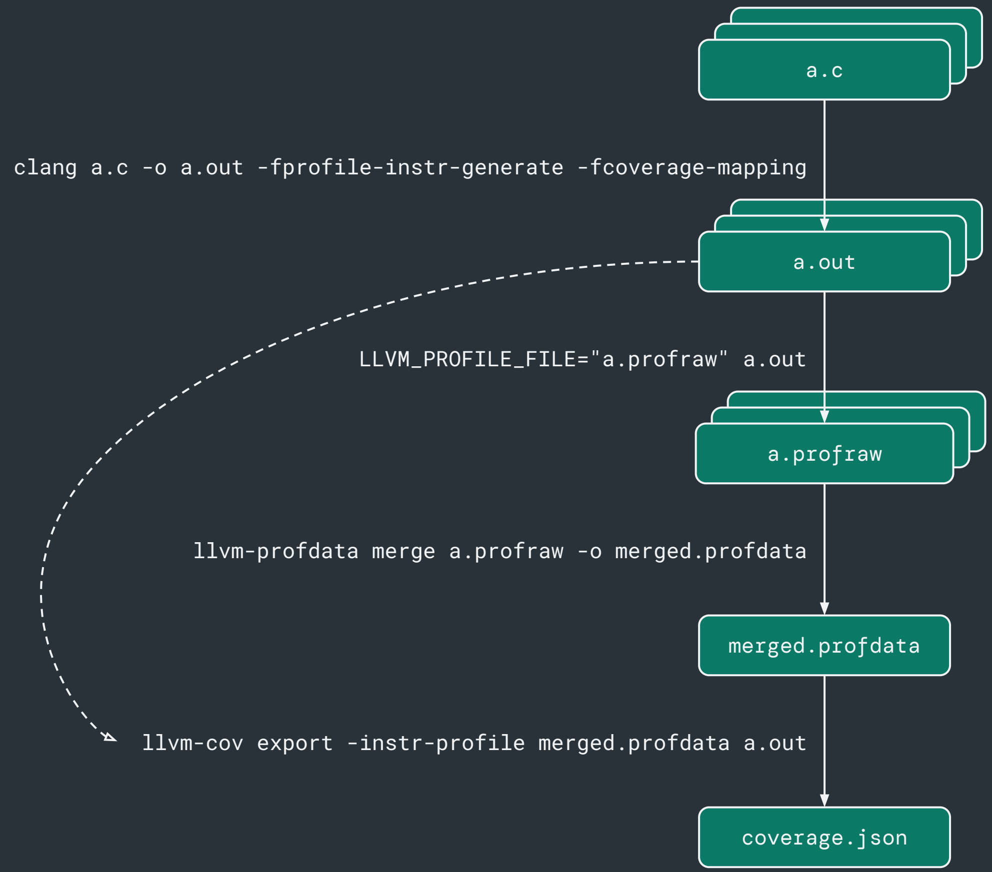
The instrumentation is applied early, before optimizations to avoid negative impact on coverage report quality.

This generates precise coverage data, but with significant performance overhead.

Source-based Code Coverage

# In Fuchsia

14,000 sources
4,500 tests
5,000 binaries
7,000 raw profiles
1GB indexed profile



# Emitting profiles with abnormal termination

The profile runtime uses an `atexit()` hook to write out the raw profile to disk.

If the process terminates abnormally, `atexit()` hooks may not be executed resulting in missing coverage.

This is a problem for tests that spawn subprocesses such as "death tests".

# Emitting profiles during abnormal termination

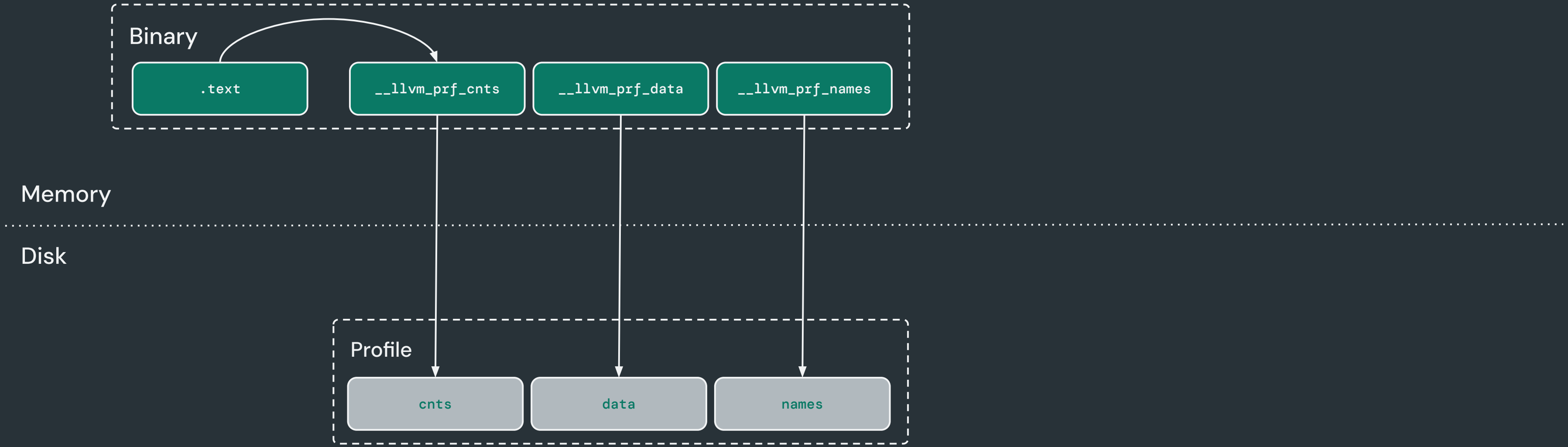
During abnormal termination, an empty profile is generated.

```
0 int main(int argc, char** argv) {
0   if (argc != 1) {
0     abort();
0   }
0   return 0;
0 }
```

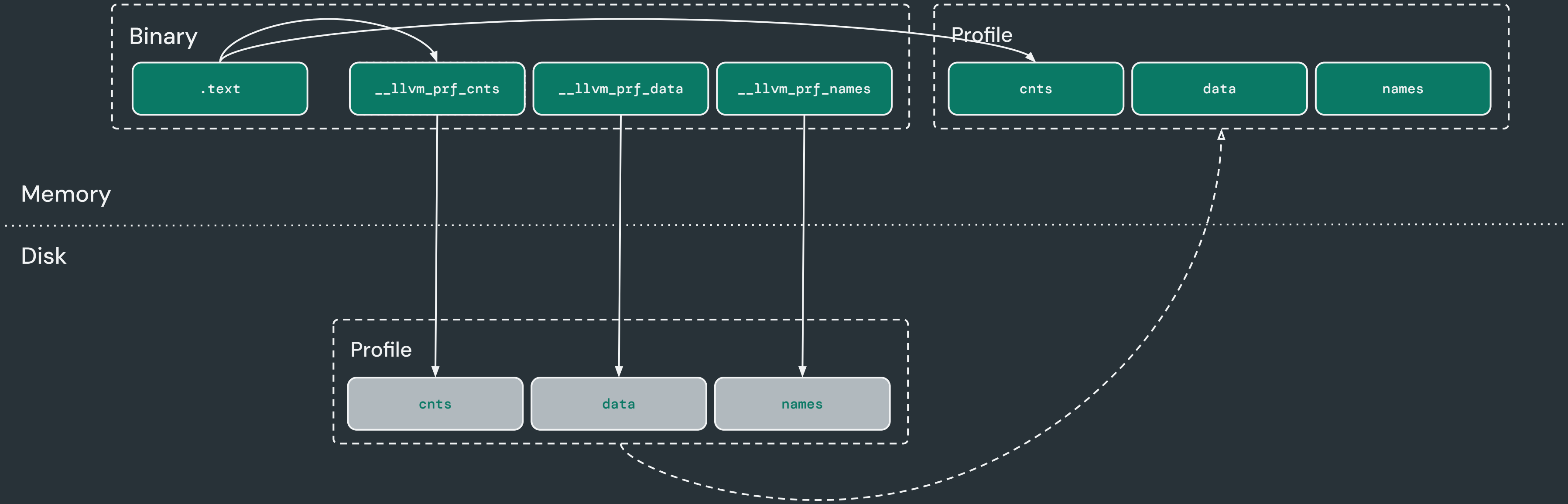
a.c

```
$ clang a.c -o a.out \
    -fprofile-instr-generate -fcoverage-mapping
$ LLVM_PROFILE_FILE="a.profraw" ./a.out LLVM DevMtg
Aborted
$ llvm-profdata merge -sparse a.profraw -o a.profdata
$ llvm-cov show ./a.out -instr-profile=a.profdata
```

```
counter update:
c = __profc[idx];
c++;
__profc[idx] = c
```



```
counter update:
c = *(&__profc[idx] + __llvm_profile_counter_bias)
c++
*(&__profc[idx] + __llvm_profile_counter_bias) = c
```




```
bias update:
__llvm_profile_counter_bias = &cnts - &__llvm_prf_cnts
```

# Using runtime counter relocation

During abnormal termination, profile is written out as expected.

```
1 int main(int argc, char** argv) {
1   if (argc != 1) {
1     abort();
0   }
0   return 0;
1 }
```

a.c

Runtime counter relocation can be enabled by a backend option and requires %c flag.

```
$ clang a.c -o a.out \
    -fprofile-instr-generate -fcoverage-mapping \
    -mllvm -runtime-counter-relocation
$ LLVM_PROFILE_FILE="a%c.profraw" ./a.out LLVM DevMtg
Aborted
$ llvm-profdata merge -sparse a.profraw -o a.profdata
$ llvm-cov show ./a.out -instr-profile=a.profdata
```

# Writing counters on-the-fly

In Fuchsia, we use runtime counter relocation by default. Since the profile is emitted at the start of the program, and the counters are updated on-the-fly, abnormal termination is no longer an issue.

Runtime counter relocation introduces a level of indirection which results in runtime overhead and increased binary size.

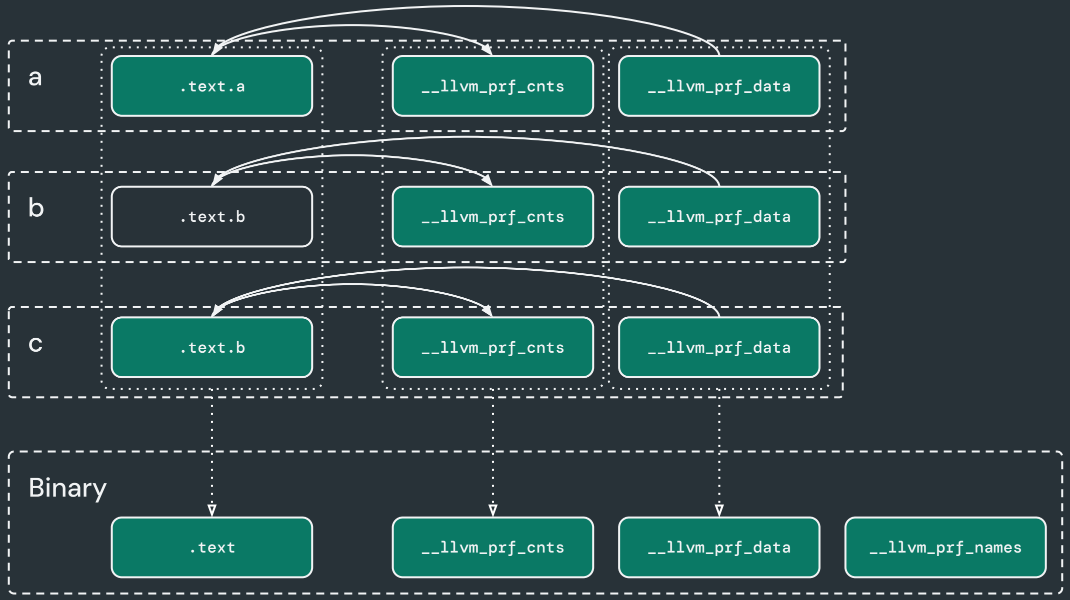
*Note that macOS uses a different approach called "continuous mode" which relies on overmapping.*

# Reducing the size of instrumented binaries

In a typical C/C++ and Rust binary, there is large number of unused functions.

In an uninstrumented build, these would be stripped by the linker `--gc-sections` feature (in ELF).

That was not possible for instrumented binaries as the metadata sections had references to the `.text` section which prevented the linker from discarding these.



a
.text.a
__llvm_prf_cnts
__llvm_prf_data
b
.text.b
__llvm_prf_cnts
__llvm_prf_data
c
.text.b
__llvm_prf_cnts
__llvm_prf_data
Binary
.text
__llvm_prf_cnts
__llvm_prf_data
__llvm_prf_names

# Support for ELF zero-flag section groups

We explored several potential solutions, we ended up introducing a new Comdat selection kind in LLVM IR: `nodeduplicate`

This is lowered to ELF zero-flag section group which is now supported by LLVM and LLD.

Addressing this issue reduced the size of instrumented binaries and generated profiles by 50% in Fuchsia.

*This approach could be used for other kinds of instrumentation that generates metadata sections.*

# Selective instrumentation

A patch typically only modifies a small subset of files.

We can significantly reduce the coverage overhead by only instrumenting the modified files.

We reuse the sanitizer special case list format to specify files/functions to allow/skip/forbid instrumentation for.

At the LLVM IR level, this translates to `noprofile` and `skipprofile` function attributes.

# Using selective instrumentation

Specify which functions and sources to allow/skip/forbid instrumentation for using the sanitizer special case list format.

```
# Only apply to frontend instrumentation.
[clang]

# Instrument function named foo.
function:foo=allow

# Instrument all source files in lib/foo.
source:lib/foo/*.c=allow

# Otherwise skip instrumentation.
default:skip
```

cov.list

`-fprofile-list` flag is used to pass the list to compiler.

```
$ clang a.c -o a.out -fprofile-list=cov.list \
    -fprofile-instr-generate -fcoverage-mapping
```

# Fuchsia coverage pipeline

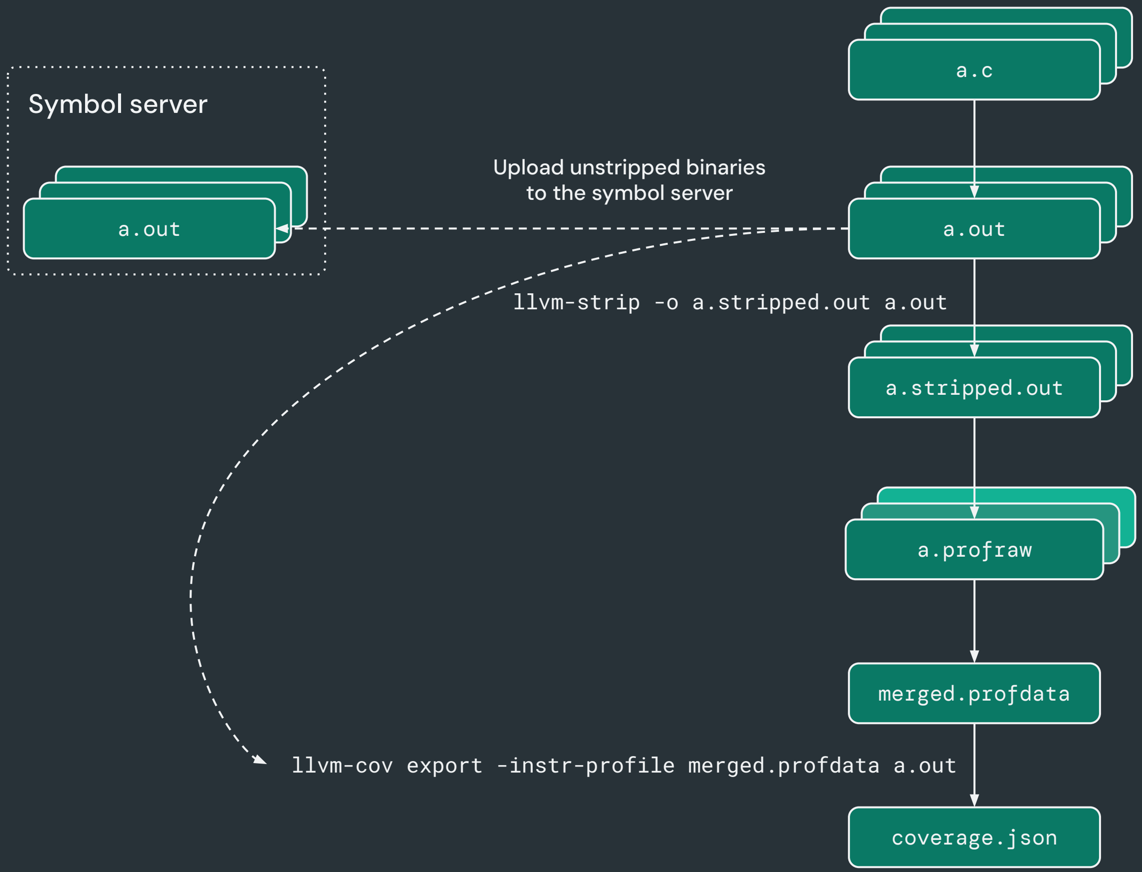
We use different machines for building, running tests and coverage post-processing.

We always strip binaries to reduce their size, and upload the unstripped binaries to symbol server.

We need to use unstripped binaries for coverage post-processing.

We need a way to associate the collected profiles with unstripped binaries during post-processing.

Building an Operating System from Scratch with LLVM



Symbol server
a.out
Upload unstripped binaries
to the symbol server
a.c
a.out
llvm-strip -o a.stripped.out a.out
a.stripped.out
a.profraw
merged.profdata
llvm-cov export -instr-profile merged.profdata a.out
coverage.json

# Embedding binary ID in profiles

Binary ID refers to the unique identifiers for binaries in different file formats.

- Build ID as a unique identifier in ELF
- `LC_UUID` as an identifier in Mach-O
- GUID used in COFF

Binary ID embedded inside the profile can be used to map the profile back to the binary that produced it.

*Note that GCC generates Build ID by default, Clang can be opted in by the `ENABLE_LINKER_BUILD_ID` CMake flag, you can also use the `--build-id` linker flag.*

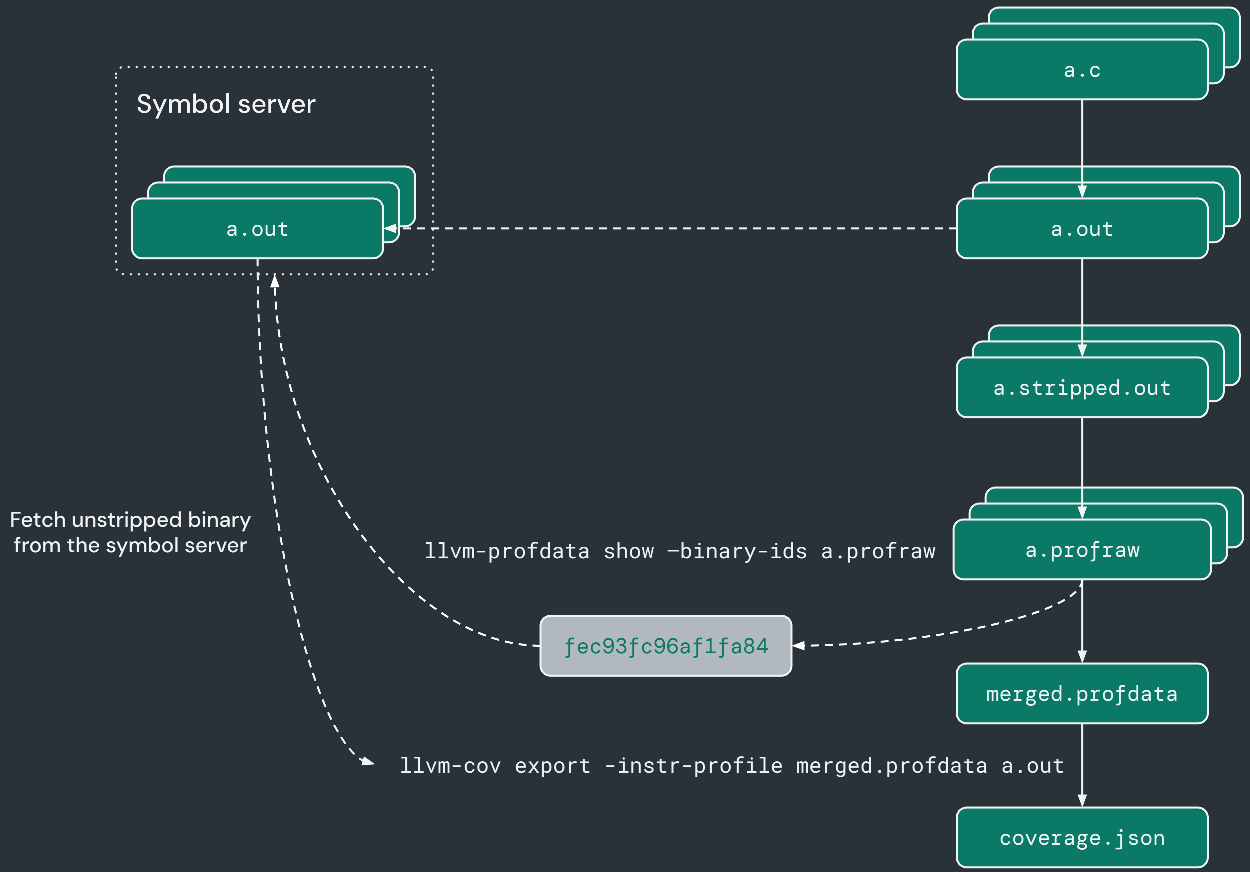
[Build ID]

# Support for binary ID in coverage

The profile runtime writes binary ID into the profile as an optional field.

`llvm-profdata` can be used to display it.

```
$ clang a.c -o a.out -Wl,--build-id \
    -fprofile-instr-generate -fcoverage-mapping
$ LLVM_PROFILE_FILE="a.profraw" ./a.out
$ llvm-profdata show --binary-ids a.profraw
Binary IDs:
02274a7974e4593e65b37d81ce602dba1b54edee
```



Symbol server
a.out
a.c
a.out
a.stripped.out
Fetch unstripped binary
from the symbol server
llvm-profdata show –binary-ids a.profraw
a.profraw
fec93fc96af1fa84
merged.profdata
llvm-cov export -instr-profile merged.profdata a.out
coverage.json

# Using binary ID in profiles

In Fuchsia, using binary ID simplified the pipeline, increased the reliability and reduced coverage post-processing time by 25%.

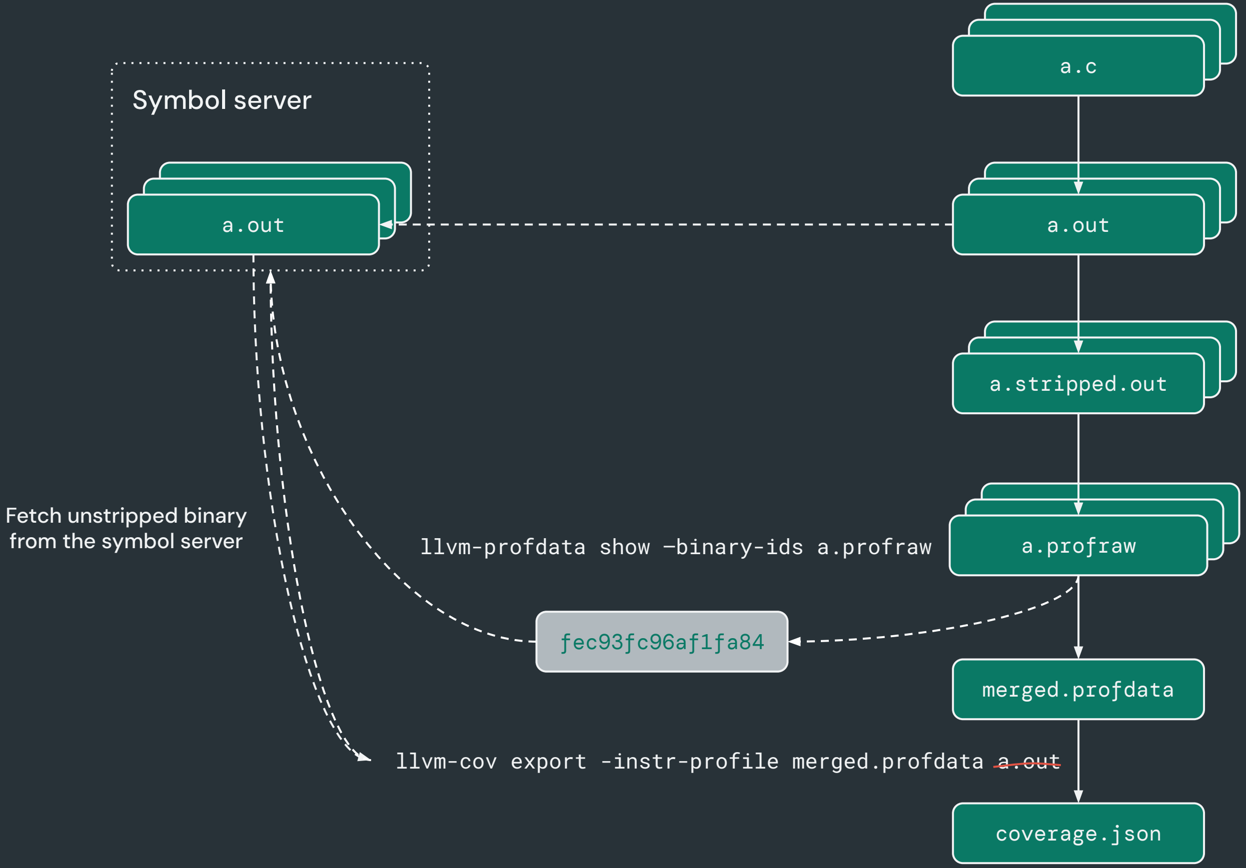
# Debuginfod support in LLVM

debuginfod is a simple HTTP API that can be used to fetch unstripped binaries by build ID.

LLVM libDebuginfod is a client/server implementation which can be easily integrated into LLVM tools.

debuginfod is already supported by `llvm-symbolize` and `llvm-objdump`.

[Introducing debuginfod, the elfutils debuginfo server]


Symbol server
a.out
a.c
a.out
a.stripped.out
a.profraw
Fetch unstripped binary
from the symbol server
llvm-profdata show -binary-ids a.profraw
fec93fc96af1fa84
merged.profdata
llvm-cov export -instr-profile merged.profdata a.out
coverage.json

# Using debuginfod for coverage

We added debuginfod support to `llvm-cov` to fetch binaries using the binary IDs embedded in the indexed profile.

This further simplified our infrastructure which already uses debuginfod for symbolization.

# Per-directory index in coverage reports

Co-mentored **Yuhao Gu** who participated in the Google Summer of Code Program (GSoC) with LLVM organization this summer.

Improved the readability of textual and HTML coverage reports by enhancing `llvm-cov`.

Enhancing llvm-cov to Generate Hierarchical Coverage Reports

# Per-directory index in coverage reports

`llvm-cov` generates a single top-level HTML index for the entire project.

For Fuchsia, this HTML has 14,000 rows and is 14MB becoming unusable in most browsers.

**Coverage Report**

**Created: 2023-10-05 02:21**

Click here for information about interpreting this report.

| Filename | Function Coverage | Line Coverage | Region Coverage | Branch Coverage |
|---|---|---|---|---|
| fuchsia-third_party-rust/library/std/src/sys/common/thread_local/fast_local.rs | 80.00% (4/5) | 96.67% (58/60) | 73.33% (11/15) | - (0/0) |
| fuchsia/build/sdk/meta/src/banjo_library.rs | 62.50% (5/8) | 93.18% (41/44) | 75.00% (12/16) | - (0/0) |
| fuchsia/build/sdk/meta/src/cc_prebuilt_library.rs | 57.14% (8/14) | 86.96% (40/46) | 73.53% (25/34) | - (0/0) |
| fuchsia/build/sdk/meta/src/cc_source_library.rs | 62.50% (5/8) | 95.08% (58/61) | 80.00% (16/20) | - (0/0) |
| fuchsia/build/sdk/meta/src/common.rs | 57.97% (40/69) | 61.47% (67/109) | 62.10% (77/124) | - (0/0) |
| fuchsia/build/sdk/meta/src/dart_library.rs | 58.33% (7/12) | 92.65% (63/68) | 74.07% (20/27) | - (0/0) |
| fuchsia/build/sdk/meta/src/data.rs | 70.00% (7/10) | 95.24% (60/63) | 75.00% (12/16) | - (0/0) |
| fuchsia/build/sdk/meta/src/documentation.rs | 62.50% (5/8) | 91.43% (32/35) | 71.43% (10/14) | - (0/0) |
| fuchsia/build/sdk/meta/src/ffx_tool.rs | 77.78% (14/18) | 89.53% (77/86) | 77.08% (37/48) | - (0/0) |
| fuchsia/build/sdk/meta/src/fidl_library.rs | 62.50% (5/8) | 93.18% (41/44) | 75.00% (12/16) | - (0/0) |
| fuchsia/build/sdk/meta/src/host_tool.rs | 71.43% (5/7) | 96.00% (48/50) | 85.71% (12/14) | - (0/0) |
| fuchsia/build/sdk/meta/src/json.rs | 64.71% (11/17) | 82.65% (81/98) | 65.67% (44/67) | - (0/0) |
| fuchsia/build/sdk/meta/src/loadable_module.rs | 71.43% (5/7) | 95.92% (47/49) | 85.71% (12/14) | - (0/0) |
| fuchsia/build/sdk/meta/src/manifest.rs | 45.00% (9/20) | 82.43% (61/74) | 63.16% (24/38) | - (0/0) |
| fuchsia/build/sdk/meta/src/metadata.rs | 80.95% (34/42) | 95.00% (437/460) | 71.15% (111/156) | - (0/0) |
| fuchsia/build/sdk/meta/src/physical_device.rs | 63.16% (12/19) | 88.33% (53/60) | 78.12% (25/32) | - (0/0) |
| fuchsia/build/sdk/meta/src/product_bundle.rs | 63.64% (28/44) | 80.90% (144/178) | 65.99% (97/147) | - (0/0) |
| fuchsia/build/sdk/meta/src/product_bundle/v1.rs | 64.04% (57/89) | 89.58% (275/307) | 77.33% (191/247) | - (0/0) |
| fuchsia/build/sdk/meta/src/product_bundle/v2.rs | 82.98% (39/47) | 93.73% (389/415) | 78.03% (245/314) | - (0/0) |
| fuchsia/build/sdk/meta/src/product_bundle_container.rs | 65.38% (17/26) | 94.66% (195/206) | 71.93% (41/57) | - (0/0) |
| fuchsia/build/sdk/meta/src/sysroot.rs | 63.64% (7/11) | 92.98% (53/57) | 84.00% (21/25) | - (0/0) |
| fuchsia/build/sdk/meta/src/testing.rs | 14.29% (1/7) | 23.53% (8/34) | 23.53% (4/17) | - (0/0) |
| fuchsia/build/sdk/meta/src/virtual_device.rs | 47.06% (8/17) | 88.04% (81/92) | 68.75% (33/48) | - (0/0) |
| fuchsia/build/sdk/meta/src/virtual_device/manifest.rs | 86.36% (38/44) | 98.03% (398/406) | 86.22% (169/196) | - (0/0) |
| fuchsia/build/sdk/meta/src/virtual_device/v1.rs | 78.72% (37/47) | 92.86% (130/140) | 84.44% (76/90) | - (0/0) |
| fuchsia/build/tools/json_merge/json_merge.cc | 100.00% (2/2) | 94.00% (47/50) | 97.06% (33/34) | 86.36% (19/22) |
| fuchsia/build/tools/json_merge/test.cc | 100.00% (15/15) | 100.00% (115/115) | 90.19% (285/316) | 50.00% (51/102) |
| fuchsia/build/tools/json_validator/src/main.rs | 62.50% (5/8) | 79.31% (46/58) | 76.92% (30/39) | - (0/0) |
| fuchsia/examples/components/config/cpp/main.cc | 100.00% (1/1) | 100.00% (12/12) | 94.12% (16/17) | 50.00% (1/2) |
| fuchsia/examples/components/config/integration_test/cpp/test.cc | 100.00% (7/7) | 100.00% (96/96) | 78.77% (167/212) | 51.56% (33/64) |
| fuchsia/examples/components/config/integration_test/rust/lib.rs | 100.00% (20/20) | 100.00% (179/179) | 96.77% (120/124) | - (0/0) |
| fuchsia/examples/components/config/rust/src/main.rs | 100.00% (5/5) | 90.00% (18/20) | 78.57% (11/14) | - (0/0) |
| fuchsia/examples/components/config_from_parent/cpp/main.cc | 100.00% (1/1) | 100.00% (11/11) | 94.12% (16/17) | 50.00% (1/2) |
| fuchsia/examples/components/config_from_parent/integration_test/cpp/test.cc | 100.00% (11/11) | 90.70% (156/172) | 47.16% (133/282) | 43.06% (31/72) |
| fuchsia/examples/components/config_from_parent/integration_test/rust/lib.rs | 88.24% (15/17) | 98.67% (148/150) | 80.81% (80/99) | - (0/0) |
| fuchsia/examples/components/config_from_parent/rust/src/main.rs | 100.00% (5/5) | 85.71% (12/14) | 78.57% (11/14) | - (0/0) |
| fuchsia/examples/components/echo/cpp/echo_component.cc | 100.00% (3/3) | 90.91% (20/22) | 90.00% (9/10) | 90.00% (9/10) |
| fuchsia/examples/components/echo/cpp/echo_unittest.cc | 100.00% (3/3) | 100.00% (15/15) | 83.33% (45/54) | 50.00% (9/18) |
| fuchsia/examples/components/echo/rust/src/main.rs | 72.22% (13/18) | 65.96% (31/47) | 68.97% (20/29) | - (0/0) |
| fuchsia/examples/components/realm_builder/cpp/sample.cc | 100.00% (11/11) | 100.00% (66/66) | 86.96% (60/69) | 50.00% (9/18) |
| fuchsia/examples/components/realm_builder/rust/src/lib.rs | 93.75% (15/16) | 94.02% (110/117) | 74.73% (68/91) | - (0/0) |
| fuchsia/examples/components/routing/cpp/echo_server/main.cc | 100.00% (3/3) | 100.00% (20/20) | 100.00% (3/3) | - (0/0) |
| fuchsia/examples/components/routing/integration_tests/cpp/echo_integration_test.cc | 100.00% (1/1) | 100.00% (9/9) | 83.78% (31/37) | 50.00% (6/12) |
| fuchsia/examples/components/routing/integration_tests/rust/src/lib.rs | 100.00% (4/4) | 100.00% (13/13) | 100.00% (9/9) | - (0/0) |
| fuchsia/examples/components/routing/rust/echo_client/src/main.rs | 80.00% (4/5) | 93.75% (15/16) | 92.86% (13/14) | - (0/0) |
| fuchsia/examples/components/routing/rust/echo_server/src/main.rs | 100.00% (7/7) | 94.29% (33/35) | 85.71% (18/21) | - (0/0) |
| fuchsia/examples/components/services/cpp/branch.cc | 100.00% (3/3) | 100.00% (58/58) | 86.61% (110/127) | 52.78% (19/36) |
| fuchsia/examples/components/services/cpp/provider.cc | 88.89% (8/9) | 77.27% (34/44) | 63.41% (26/41) | 33.33% (4/12) |
| fuchsia/examples/components/services/rust/src/branch.rs | 100.00% (11/11) | 100.00% (77/77) | 100.00% (52/52) | - (0/0) |
| fuchsia/examples/components/services/rust/src/provider.rs | 83.33% (10/12) | 87.14% (61/70) | 71.21% (47/66) | - (0/0) |
| fuchsia/examples/components/storage/src/main.rs | 58.62% (17/29) | 80.16% (101/126) | 67.78% (61/90) | - (0/0) |
| fuchsia/examples/components/subpackages/src/lib.rs | 100.00% (10/10) | 100.00% (82/82) | 100.00% (60/60) | - (0/0) |
| fuchsia/examples/cpp/synchronization_checker/main.cc | 100.00% (6/6) | 95.24% (100/105) | 83.96% (89/106) | 50.00% (15/30) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/fizzbuzz/main.cc | 100.00% (6/6) | 100.00% (53/53) | 96.77% (30/31) | 92.86% (13/14) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_1/reverser.cc | 33.33% (1/3) | 37.50% (6/16) | 66.67% (4/6) | 100.00% (2/2) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_1/reverser_unittests.cc | 100.00% (2/2) | 100.00% (7/7) | 100.00% (2/2) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_1/tests/integration_test.cc | 50.00% (2/4) | 83.33% (10/12) | 50.00% (2/4) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_2/reverser.cc | 100.00% (4/4) | 100.00% (35/35) | 100.00% (10/10) | 100.00% (2/2) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_2/reverser.h | 100.00% (1/1) | 100.00% (4/4) | 100.00% (1/1) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_2/reverser_unittests.cc | 100.00% (4/4) | 100.00% (22/22) | 95.00% (19/20) | 50.00% (3/6) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_2/tests/integration_test.cc | 75.00% (6/8) | 95.00% (38/40) | 87.80% (72/82) | 56.25% (18/32) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_3/reverser.cc | 100.00% (4/4) | 100.00% (33/33) | 100.00% (10/10) | 100.00% (2/2) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_3/reverser.h | 100.00% (1/1) | 100.00% (4/4) | 100.00% (1/1) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_3/reverser_unittests.cc | 100.00% (4/4) | 100.00% (22/22) | 95.00% (19/20) | 50.00% (3/6) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_3/tests/integration_test.cc | 75.00% (6/8) | 95.00% (38/40) | 87.80% (72/82) | 56.25% (18/32) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_4/reverser.cc | 100.00% (4/4) | 100.00% (33/33) | 100.00% (10/10) | 100.00% (2/2) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_4/reverser_unittests.cc | 100.00% (10/10) | 100.00% (93/93) | 88.04% (243/276) | 50.00% (45/90) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_4/tests/integration_test.cc | 75.00% (6/8) | 95.00% (38/40) | 87.80% (72/82) | 56.25% (18/32) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_5/main.cc | 100.00% (3/3) | 100.00% (33/33) | 92.31% (24/26) | 50.00% (2/4) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_5/reverser.cc | 100.00% (4/4) | 100.00% (33/33) | 100.00% (10/10) | 100.00% (2/2) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_5/reverser_unittests.cc | 100.00% (10/10) | 100.00% (93/93) | 88.04% (243/276) | 50.00% (45/90) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/part_5/tests/integration_test.cc | 81.82% (9/11) | 97.06% (99/102) | 88.49% (123/139) | 57.41% (31/54) |
| fuchsia/examples/diagnostics/inspect/codelab/cpp/testing/integration_test.cc | 100.00% (2/2) | 100.00% (33/33) | 100.00% (4/4) | 100.00% (2/2) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/fizzbuzz/src/main.rs | 81.25% (13/16) | 91.67% (77/84) | 78.26% (36/46) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/part_1/src/main.rs | 0.00% (0/9) | 0.00% (0/30) | 0.00% (0/30) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/part_1/src/reverser.rs | 60.00% (6/10) | 47.22% (17/36) | 50.00% (14/28) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/part_1/tests/integration_test.rs | 100.00% (8/8) | 100.00% (16/16) | 84.62% (22/26) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/part_2/src/main.rs | 66.67% (6/9) | 86.00% (43/50) | 63.33% (19/30) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/part_2/src/reverser.rs | 100.00% (12/12) | 100.00% (73/73) | 91.18% (31/34) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/part_2/tests/integration_test.rs | 100.00% (8/8) | 100.00% (20/20) | 83.33% (30/36) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/part_3/src/main.rs | 85.71% (6/7) | 95.24% (40/42) | 82.61% (19/23) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/part_3/src/reverser.rs | 100.00% (12/12) | 100.00% (69/69) | 91.18% (31/34) | - (0/0) |
| fuchsia/examples/diagnostics/inspect/codelab/rust/part_3/tests/integration_test.rs | 100.00% (8/8) | 100.00% (20/20) | 83.33% (30/36) | - (0/0) |

# Per-directory index in coverage reports

`llvm-cov` can generate per-directory index with `-show-directory-coverage`

[Support directory layout in coverage reports]

## Coverage Report (/b/s/w/ir/x/w/fuchsia/)

**Created: 2023-10-05 22:15**

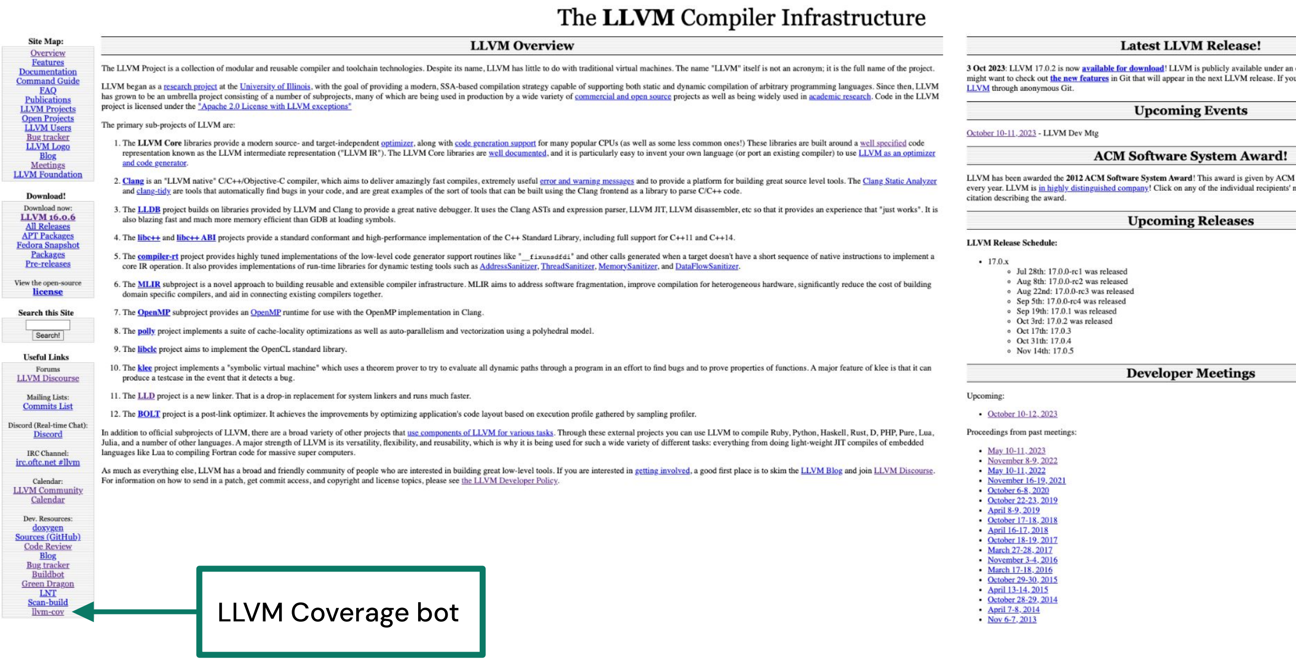
Click here for information about interpreting this report.

| Filename | Function Coverage | Line Coverage | Region Coverage | Branch Coverage |
|---|---|---|---|---|
| build/ | 68.27% (426/624) | 90.66% (3087/3405) | 77.29% (1674/2166) | 56.45% (70/124) |
| examples/ | 84.83% (1381/1628) | 87.45% (10854/12412) | 70.33% (12555/17851) | 48.17% (1760/3654) |
| out/not-default/ | 17.93% (77347/431366) | 45.88% (1443421/3146152) | 41.53% (462908/1114557) | 50.74% (69612/137204) |
| prebuilt/third_party/ | 4.77% (439/9211) | 9.10% (3684/40503) | 13.80% (1527/11067) | 40.39% (605/1498) |
| sdk/ | 83.89% (9562/11398) | 85.64% (74647/87168) | 71.01% (203176/286143) | 54.14% (33338/61580) |
| src/ | 76.36% (221463/290015) | 81.80% (2026752/2477799) | 69.41% (2713333/3909112) | 48.72% (434437/891734) |
| third_party/ | 26.85% (53031/197483) | 24.58% (528142/2148916) | 29.90% (629325/2104658) | 19.63% (169375/862706) |
| tools/ | 81.62% (8837/10827) | 88.29% (106831/120995) | 50.26% (203514/404937) | 42.19% (30173/71520) |
| zircon/ | 71.44% (17636/24688) | 62.31% (178548/286534) | 52.05% (523156/1005128) | 42.30% (87794/207544) |
| Totals | 39.92% (390125/977244) | 52.57% (4375981/8323900) | 53.65% (4751186/8855644) | 36.97% (827164/2237564) |

**Generated by llvm-cov -- llvm version 18.0.0git**

# Per-directory index in coverage reports

This feature is also enabled for LLVM coverage reports.




**Coverage Report (/Users/buildslave/jenkins/workspace/coverage/llvm-project/)**

**Created: 2023-10-07 09:23**

Click here for information about interpreting this report.

| Filename | Function Coverage | Line Coverage | Region Coverage | Branch Coverage |
|---|---|---|---|---|
| clang/ | 89.25% (56326/63113) | 88.28% (704033/797518) | 76.05% (578466/760649) | 76.29% (387845/508360) |
| lldb/ | 70.43% (18308/25995) | 59.53% (185200/311089) | 58.65% (118152/201447) | 47.19% (62080/131558) |
| llvm/ | 86.83% (91053/104860) | 86.41% (1214804/1405799) | 81.07% (900153/1110362) | 78.71% (629281/799514) |
| Totals | 85.42% (165688/193972) | 83.68% (2104050/2514436) | 77.05% (1596772/2072464) | 74.97% (1079206/1439436) |

Generated by llvm-cov -- llvm version 18.0.0git

# Q&A

There are of opportunities for further improvements that we would like to explore in the future.

If you're interested in collaborating on the ideas listed on this slide, please reach out.

**Support for boolean counters to reduce runtime overhead**
For coverage, we only need to know if region was executed

**Omit the unnecessary sections from binaries and profiles**
Sections other than counters can be stripped for coverage

**Support string merging for `__llvm_prf_names`**
This would enable deduplication resulting in 10x reduction

**Make `__llvm_prf_data` position independent**
Avoid per-function dynamic relocation and allow sharing

**Avoid the use of indirection for runtime counter relocation**
This requires assistance from linker and dynamic linker