October 12, 2023



Using Clang's source-based code coverage at scale

+90 -0

Code coverage in Fuchsia

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

```
#include "backtrace.h"
 5
 6
   #include "threads_impl.h"
   namespace __libc_sanitizer {
 9
  size_t BacktraceByFramePointer(cpp20::span<uintptr_t> pcs) {
     struct FramePointer {
12
13
       const FramePointer* fp;
14
       uintptr_t pc;
15
     };
16
     auto on_stack = [&stack = __pthread_self()->safe_stack](const FramePointer* fp) -> bool {
17
18
       uintptr_t address = reinterpret_cast<uintptr_t>(fp);
19
       return address >= reinterpret_cast<uintptr_t>(stack.iov_base) &&
20
              address < reinterpret_cast<uintptr_t>(stack.iov_base) + stack.iov_len;
     };
21
22
23
     uintptr_t ra = reinterpret_cast<uintptr_t>(__builtin_return_address(0));
24
     auto fp = reinterpret_cast<const FramePointer*>(__builtin_frame_address(0));
25
     size_t i = 0;
26
     while (i < pcs.size() && on_stack(fp) && fp->pc != 0) {
       if (i == 0 && fp->pc != ra) {
27
         pcs[i++] = ra;
28
29
       } else {
30
         pcs[i++] = fp->pc;
31
         fp = fp - > fp:
32
33
34
     if (i == 0 && i < pcs.size()) {</pre>
35
       pcs[i++] = ra;
36
37
38
     return i:
39
   #if __has_feature(shadow_call_stack)
41
42
43
   namespace {
44
```

Code coverage in Fuchsia

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

Files Outline <1	☆ backtrace.cc
Repository root inttypes libc++-stub math pthread sanitizers BUILD.gn	<pre>1 // Copyright 2021 The Fuchsia Authors. All rights reserved. 2 // Use of this source code is governed by a BSD-style license that can be 3 // found in the LICENSE file. 4 5 #include "backtrace.h" 6 7 #include <lib arch="" backtrace.h=""> 8 9 #include "threads_impl.h" 10 11 namespacelibc_sanitizer {</lib></pre>
 _asan_early_init.c _sanitizer_fast_backtrac asan-stubs.c backtrace-tests.cc 	<pre>12 13 size_t BacktraceByFramePointer(cpp20::span<uintptr_t> pcs) { 14 struct IsOnStack { 15 bool operator()(const arch::CallFrame* fp) const { 16 const iovec& stack =pthread_self()->safe_stack; 17 if (stack.iov_len < sizeof(*fp)) [(unlikely]] { 10 const iovecwell =pthread_self() =pth</uintptr_t></pre>
 backtrace.cc backtrace.h debugdata.cc fuchsia-io-constants.h hooks.c hwasan-stubs.cc hwasan-stubs.h log.c memory-snapshot.cc sancov-stubs.h sanitizer-stubs.h ubsan-stubs.cc scudo setjmp stdio 	<pre>18 // This should be impossible, but assume nothing in a critical 19 // trior-reporting path since this might be used after clobberation. 20 return false; 21 } 22 const uintptr_t base = reinterpret_cast<uintptr_t>(stack.iov_base); 23 const uintptr_t frame = reinterpret_cast<uintptr_t>(fp); 24 return frame >= base && frame - base <= stack.iov_len - sizeof(*fp); 25 } 26 }; 27 using FpBacktrace = arch::FramePointerBacktrace<isonstack>; 28 29 return arch::StoreBacktrace(FpBacktrace::BackTrace(), pcs,builtin_return_address(0)); 30 } 31 32 #ifhas_feature(shadow_call_stack) 34 size_t BacktraceByShadowCallStack(cpp20::span<uintptr_t> pcs) { 5 const iovec& shadow_call_stack_block =pthread_self()->shadow_call_stack; 36 return arch::StoreBacktrace(37 arch::ShadowCallStackBacktrace{ 38 { static_cast<const uintptr_t*="">(shadow_call_stack_block.iov_base), 39 shadow_call_stack_block.iov_len / sizeof(uintptr_t)), 40 arch::GetShadowCallStackPointer()), 41 pcs,builtin_return_address(0)); 42 } 43 bite_t const iovecs(tastack_block.iov_len / sizeof(uintptr_t)), 44 arch::GetShadowCallStackPointer()), 45 pcs,builtin_return_address(0)); 45 bite_tast_const (intptr_t)), 46 bite_tast_const (intptr_t)), 47 arch::GetShadowCallStackPointer()), 48 bite_tast_const (intptr_t)), 49 bite_tast_const (intptr_t)), 40 bite_tast_const (intptr_t)), 41 bite_tast_const (intptr_t)), 42 } 42 }</const></uintptr_t></isonstack></uintptr_t></uintptr_t></pre>
 stdlib string stubs test zircon 	<pre>43 44 #endif //has_feature(shadow_call_stack) 45 46 } // namespacelibc_sanitizer</pre>

fuchsia/fuchsia 👻 🗲 📴 main 💌 🗲 zircon/system/ulib/c/sanitizers/backtrace.cc

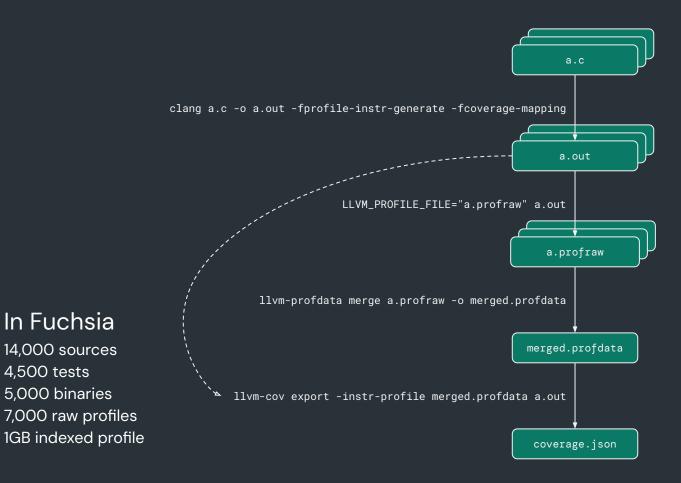
Clang source-based code coverage

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



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.

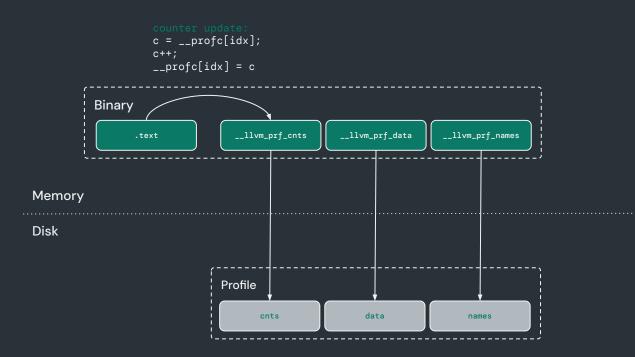
This is a problem for tests that spawn subprocesses such as "death tests".

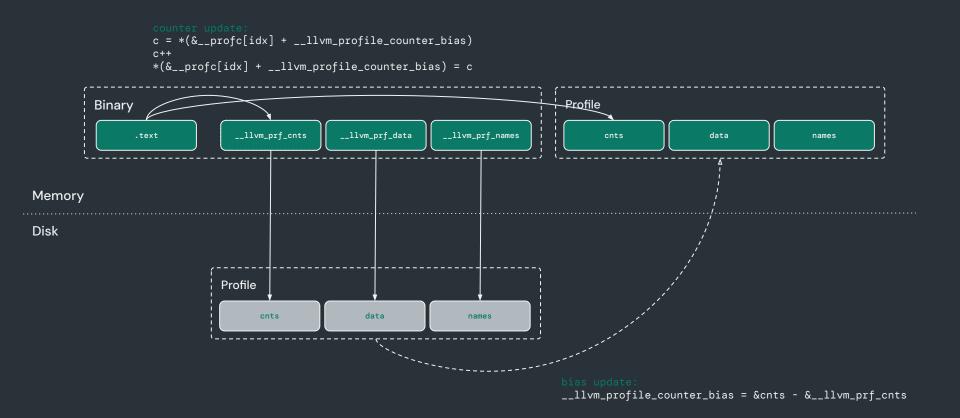
Emitting profiles during abnormal termination

During abnormal termination, an empty profile is generated.

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

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





Using runtime counter relocation

During abnormal termination, profile is written out as expected.

1 int main(int argc, char** argv) { **if** (argc != 1) { 1 abort(); 1 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.

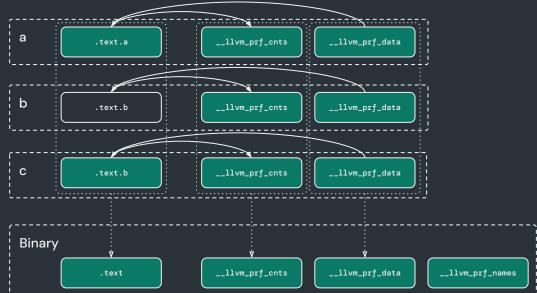
Note that macOS uses a different approach called "continuous mode" which relies on overmapping.

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.



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

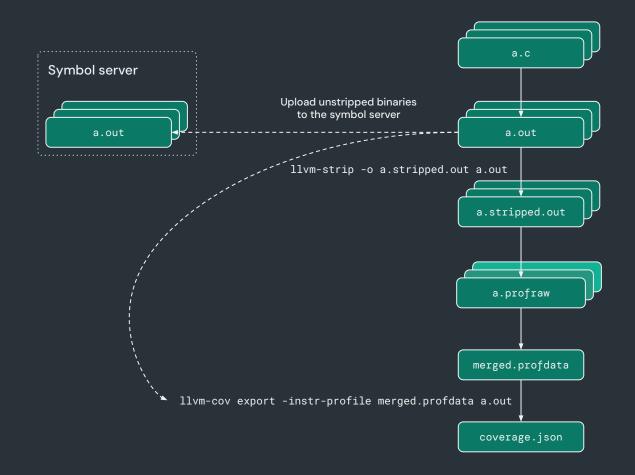
We use different machines for building, running tests and coverage post-processing.

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



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.

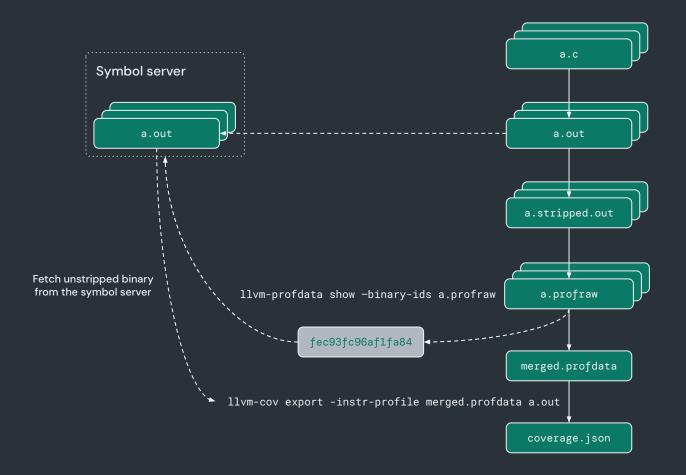


Support for binary ID in coverage

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



Using binary ID in profiles

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

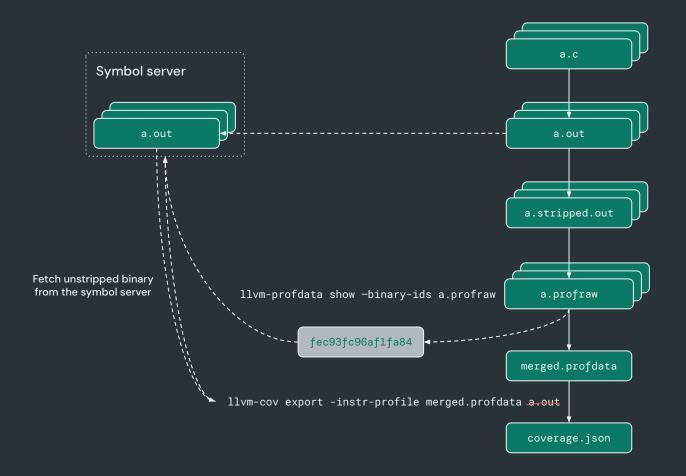
Debuginfod support in LLVM

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



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 <u>Google</u> <u>Summer of Code Program</u> (GSoC) with LLVM organization this summer.

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

Enhancing Ilvm-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 Covera
fuchsia-third_warty_rust/library/std/src/sys/commos/thread_local/fast_local.rs	80.00% (4/5)	96.678 (58/60)	73.338 (11/15)	- (0/0)
websis/build/sdk/gets/src/banio_library.rs	62.50% (5/8)	93.108 (41/44)	75.008 (12/16)	- (0/0)
schels/build/sdk/msta/src/cc_probuilt_library.rs	57.14% (8/14)	86.961 (40/46)	73.538 (25/34)	- (0/0)
chsia/build/adk/meta/src/cc_source_library.rs	62.503 (5/8)	95.005 (58/61)	80,005 (16/20)	- (0/0)
chaia/build/adk/meta/arc/compon.ra	57.97% (40/69)	61.478 (67/109)	62.10% (77/124)	- (0/0)
schsla/bulld/sdk/ssta/src/dart library.rs	58,333 (7/12)	92.655 (63/68)	74,078 (20/27)	- (0/0)
wchsia/build/sdk/geta/src/data.rs	70,00% (7/10)	95,248 (60/63)	75,008 (12/16)	- (0/0)
websia/build/sdk/reta/src/documentation.rs	62.505 (5/8)	91,435 (32/35)	71,438 (10/14)	= (0/0)
holia/hulid/sk/treta/scs/Zfs_tool.ss	77.78% (14/18)	89.535 (77/86)	77.08% (37/48)	- (0/0)
adhaia/buila/adk/meta/acc/fidl_library.ra	62.50% (5/8)	93.18% (41/44)	75.00% (12/16)	- (0/0)
ochsia/kuild/ski/meta/src/host_tool.rs	71.43% (5/7)	96.00% (48/50)	85.71% (12/14)	- (0/0)
schsis/Build/sdk/meta/src/ison.re	64.71% (11/17)	82.65% (81/98)	65.67% (44/67)	- (0/0)
schnis/Build/sdk/meta/src/losdable_module.rz	71.43% (5/7)	95.92% (47/49)	85.718 (12/14)	- (0/0)
bohsia/build/sdk/seta/src/manifest.ra	45.00% (9/20)	82.43% (61/74)	63.16% (24/38)	- (0/0)
uchsia/Build/sdk/meta/src/metadata.ra	80.95% (34/42)	95.00% (437/460)	71.15% (111/156)	- (0/0)
websia/Build/sdk/meta/arc/physical_device.rs	63.168 (12/19)	88.33% (53/60)	78.128 (25/32)	- (0/0)
vehsia/Build/sdk/meta/sre/oroduet. bundle.rs	63.64% (28/44)	80.908 (144/178)	65.998 (97/147)	- (0/0)
schsta/hulld/sdk/meta/scc/scoduct_build/v1.ze	64.048 (57/89)	89.585 (275/307)	77.33% (191/247)	- (0/0)
achain/will/ski/meta/acc/acabiti.humile/v2.rg	82.98% (39/47)	93.738 (389/415)	78.03% (245/314)	- (0/0)
Annual and the March Ma	65.38% (17/26)	94.668 (195/206)	71.938 (41/57)	
				- (0/0)
bohais/Build/ndk/meta/arc/ayaroot.xa	63.64% (7/11)	92.90% (53/57)	84.00% (21/25)	- (0/0)
xxhia/kuid/wdk/weta/src/tenting.rz	14.29% (1/7)	23.53% (8/34)	23.53% (4/17)	- (0/0)
bohala/build/idk/meta/src/virtual_device.re	47.06% (8/17)	88.04% (81/92)	68.75% (33/48)	- (0/0)
behsis/build/sdk/seta/src/virtual_device/manifest.rs	86.36% (38/44)	98.038 (398/406)	86.228 (169/196)	- (0/0)
uchsia/Build/sdk/meta/src/virtual_device/vl.rs	78.728 (37/47)	92.86% (130/140)	84.44% (76/90)	- (0/0)
uchsia/build/tools/ison_merge/ison_merge.cc	100.00% (2/2)	94.00% (47/50)	97.06% (33/34)	86.36% (19/3
wohsia/kuild/tools/ison_merge/test.cc	100.00% (15/15)	100.008 (115/115)	90.198 (285/316)	50.00% (51/
schsis/build/tools/json_validstor/src/ssin.rs	62.50% (5/8)	79.318 (46/58)	76.928 (30/39)	- (0/0)
uchała/examples/component/conflut/conf	100.008 (1/1)	100.008 (12/12)	94.128 (16/17)	50.008 (1/2
anna / sumples/components/conluctors/logina.co	100.00% (7/7)	100.008 (96/96)	78.778 (167/212)	51.568 (33/
schii/scmples/cemponts/confi/itateration_tes/cemp/thi.ce obhi/scmples/cemponts/confi/itateration_tes/cemp/thi.ce	100.00% (7/7)		78.77% (167/212) 96.77% (120/124)	51.56% (33/)
		100.00% (179/179)		
xxhala/examples/components/configTust/src/main.cs	100.00% (5/5)	90.00% (18/20)	78.57% (11/14)	- (0/0)
schala/examples/components/config_from_parent/cpp/mein.ec	100.00% (1/1)	100.00% (11/11)	94.128 (16/17)	50.00% (1/2)
inthis/examples/components/config_from_parent/integration_test/opp/test.co	100.00% (11/11)	90.70% (156/172)	47.16% (133/282)	43.06% (31/
ichsia/examples/components/config_from_parent/integration_test/rost/lib.rs	88.24% (15/17)	98.678 (148/150)	80.81% (80/99)	- (0/0)
uchsis/examples/components/config_from_parent/rust/src/main.rs	100.00% (5/5)	85.718 (12/14)	78.578 (11/14)	- (0/0)
wohsis/examples/components/echo/component.co	100.00% (3/3)	90.918 (20/22)	90.00% (9/10)	90.008 (9/1)
schsis/examples/components/echo/cops/echo_unittest.co	100.00% (3/3)	100.00% (15/15)	83.238 (45/54)	50.008 (9/1
vohals/examples/components/echo/rust/src/main_rs	72.22% (13/18)	65.96% (31/47)	68.97% (20/29)	- (0/0)
Vorhala /examples/coopponents/reals_builder/coop/example.co	100.00% (11/11)	100.00% (66/66)	86.96% (60/69)	50.00% (9/1
source and a second	93.75% (15/16)	94.028 (110/117)	74,738 (68/91)	- (0/0)
South and Components / Components / Company / Components / Company / Components / Company / Components / Company / C	100,009 (3/3)	100.00% (20/20)	100,00% (3/3)	- (0/0)
				50,008 (6/1
initia/examples/compinents/routiny/integration_tests/cop/etho_integration_test.co	100.00% (1/1)	100.00% (9/9)	83.78% (31/37)	
indula/examples/components/routiny/integration_tests/rout/src/lib.rs	100.00% (4/4)	100.00% (13/13)	100.00% (9/9)	- (0/0)
hebsis/examples/components/routiny/rust/echo_client/src/main.rs	80.00% (4/5)	93.75% (15/16)	92.86% (13/14)	- (0/0)
bchsia/examples/components/routing/rust/echo_server/src/main.rs	100.00% (7/7)	94.298 (33/35)	85.71% (18/21)	- (0/0)
bohsis/examples/components/services/cop/branch.co	100.00% (3/3)	100.00% (58/58)	86.61% (110/127)	\$2.78% (19/
bohais/examples/components/services/cpp/provider.cc	88.895 (8/9)	77.278 (34/44)	63.418 (26/41)	33.338 (4/1
ucheis/components/services/rust/src/branch.rs	100.00% (11/11)	100.00% (77/77)	100.00% (52/52)	- (0/0)
uchaia/examples/components/services/rust/src/provider.rs	83,335 (10/12)	87.145 (61/70)	71,218 (47/66)	- (0/0)
uchsia/examples/components/storage/src/msin.rg	58,628 (17/29)	80,16% (101/126)	67.788 (61/90)	- (0/0)
techs i a/examples/components/sites/cappa/src/11b.rs	100,00% (10/10)	100,005 (82/82)	100,00% (60/60)	- (0/0)
	100.00% (6/6)	95.24% (100/105)	83,965 (89/106)	50.00% (15/
iunhaid canang bed/eps/apuncheniaation_theokec/main.co			83.968 (89/108) 96.778 (30/31)	92,868 (13/
uchsia/example/dispostics/ispost/codelab/exp/fizzu/zz/main.cc	100.00% (6/6)	100.00% (53/53)		
sehsia/examples/diagnostics/inspect/codelab/opp/part_l/coverser.co	33.33% (1/3)	37.50% (6/16)	66.67% (4/6)	100.00% (2/2
schsia/examples/disputsics/inspect/codelab/cpp/part_l/reverser_unitests.cc	100.00% (2/2)	100.00% (7/7)	100.00% (2/2)	- (0/0)
xchria/examples/dispositics/imspect/codelab/cpg/part_l/tests/imtegration_test.cc	50.00% (2/4)	83.33% (10/12)	50.00% (2/4)	- (0/0)
schele/imamples/diamontics/imapect/codelab/cpp/part_2/reverser.cc	100.00% (4/4)	100.00% (35/35)	100.00% (10/10)	100.00% (2/2
xchsia/examples/diaquostics/inspect/codelab/epp/part_2/reverser.h	100.00% (1/1)	100.00% (4/4)	100.00% (1/1)	- (0/0)
wchsia/examples/diagnostics/inapect/codelab/cpp/part_2/reverser_unitiests.cc	100.00% (4/4)	100.00% (22/22)	95.00% (19/20)	50.00% (3/6
schsla/examples/dlamostics/inspect/codelab/eps/part_2/tests/integration_test.cc	75.001 (6/8)	95.00% (38/40)	87.80% (72/82)	56.250 (18/
schsia/examples/diaquostics/inspect/codelab/epp/part_J/reverser.cc	100.00% (4/4)	100.008 (33/33)	100.008 (10/10)	100.00% (2/2
uthsia/examples/diamostics/inspect/codelab/cpm/wst_l/reverser.h	100.00% (1/1)	100.008 (4/4)	100.00% (1/1)	- (0/0)
schlaften leg/dispect/collapse	100.00% (4/4)	100.00% (22/22)	95.00% (19/20)	50.00% (3/6
SOBLAY-SHARPLERY/LANDERLEY/SHAPEST CODELAR/OPP/SHX_J/TOPPETER_SALXEDILG.CO SOBLAY-SHAPLERY/LANDERLEY/SOBLAY/OPP/SHX_J/TOPPETER_SALXEDILG.CO	75.00% (4/4)	95.005 (38/40)	95.00% (19/20) 87.80% (72/82)	56.258 (18/
xchsia/examples/dispostics/inspect/codelsh/cpp/patt_i/cpresser.cc	100.00% (4/4)	100.00% (33/33)	100.00% (10/10)	100.00% (2/2
uchsia/examples/diagnostics/inspect/codelab/cpp/part_s/reverser_unittests.cc	100.00% (10/10)	100.00% (93/93)	88.04% (243/276)	50.00% (45/
xebsis/examples/dispositics/inspect/codelab/cpp/part_5/tests/integration_test.cc	75.00% (6/8)	95.00% (38/40)	87.80% (72/82)	56.25% (18/
schela/examples/diagnostics/inspect/codelab/epp/part_5/main.cc	100.00% (3/3)	100.00% (33/33)	92.31% (24/26)	50.00% (2/4
uchsia/examples/diaquostics/inspect/codelab/epp/part_3/reverser.cc	100.00% (4/4)	100.00% (33/33)	100.00% (10/10)	100.00% (2/2)
uchsia/examples/diaquostics/inspect/codelab/epp/part_5/reverser_unittests.cc	100.00% (10/10)	100.00% (93/93)	88.04% (243/276)	50.00% (45/
schsia/examples/diamostics/inspect/codelab/epp/part_3/tests/integration_test.co	81.825 (9/11)	97.068 (99/102)	88.498 (123/139)	57.418 (31/
oohiis/examples/dismostics/imsect/codelab/opp/testing/interation_test.oc	100.009 (2/2)	100.005 (33/33)	100.005 (4/4)	100.008 (2/2
Annual ACCAMPANDA AND AND AND AND AND AND AND AND AND	81.25% (13/16)	91.678 (77/84)	78.26% (36/46)	- (0/0)
enhalszwagiekszagiekszagiekszagiekszegy a szerzentetetetetetetetetetetetetetetetetetet	0.008 (0/9)	91.676 (77/84)	/8.26% (36/46) 0.00% (0/30)	- (0/0)
schsis/examples/dispositics/inspect/codelsb/rwit/part_l/src/reverser.rs	60.00% (6/10)	47.22% (17/36)	50.00% (14/28)	- (0/0)
schsis/examples/dispositics/inspect/codelsb/rust/part_l/tests/integration_test.rz	100.00% (8/8)	100.00% (16/16)	84.62% (22/26)	- (0/0)
chola/examples/diaquostics/inspect/codelab/rust/part_Z/src/main.rs	66.67% (6/9)	86.00% (43/50)	63.33% (19/30)	- (0/0)
uchsia/examples/diagnostics/inspect/codelab/rust/part_2/grc/reverser.rs	100.00% (12/12)	100.00% (73/73)	91.18% (31/34)	- (0/0)
	100.005 (8/8)	100.00% (20/20)	83.338 (30/36)	- (0/0)
uchsia/examples/diaquostics/inspect/codelab/rust/yart_2/testa/integraties_test.rs	100.00% (8/8)			
	85.718 (6/7)	95.248 (40/42)	82.618 (19/23)	
fachsiafannolmafilamenticeliamenticeliamenticeliamentamente in terrateriamente in terrateriamente in terrateriamente in terrateriamente international internat				- (0/0)

Per-directory index in coverage reports

llvm-cov can generate
per-directory index with
-show-directory-cove
rage

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/	68.278 (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)	45.00% (1443421/3146152)	41.53% (462908/1114557)	50.74% (69612/137204)
prebuilt/third_party/	4.77% (439/9211)	9.101 (3684/40503)	13.80% (1527/11067)	40.39% (605/1498)
adk/	83.89% (9562/11398)	85.64% (74647/87168)	71.01% (203176/286143)	54.14% (33338/61580)
are/	76.36% (221463/290015)	81.80% (2026752/2477799)	69.41% (2713333/3909112)	48.72% (434437/891734)
third_party/	26.85% (53031/197493)	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	39.928 (390125/977244)	52.57% (4375981/8323900)	53.65% (4751186/8855644)	36.978 (827164/2237564

Generated by llvm-cov - llvm version 18.0.0git

Support directory layout in coverage reports

The LLVM Compiler Infrastructure

Latest LLVM Release!

Per-directory index in coverage reports

This feature is also enabled for LLVM coverage reports.

Overview	LLVM Overview	Latest LLVM Release!
Eeatures Documentation Command Guide EAQ	The LUM Project is a collection of modular and resuble compiler and toolchain technologies. Despite its name, LUM has little to do with traditional virtual machines. The name "LUM" itself is not an acrossym; it is the full name of the project. LUM began as a <u>research project</u> at the <u>University of Illinsis</u> , with the goal of providing a modern. SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then, LUM	3 Oct 2023: LLVM 17.0.2 is now available for download! LLVM is publicly available under an might want to check out the new features in Git that will appear in the next LLVM release. If yo LLVM through anonymous Git.
Publications LLVM Projects	has grown to be an umberalla project consisting of a number of subprojects, many of which are being used in production by a wide variety of <u>commercial and open source</u> projects as well as being widely used in <u>academic research</u> . Code in the LLVM project is licensed under the <u>Apache 20 License with LLVM exceptions</u> .	Upcoming Events
Open Projects LLVM Users Bug tracker	The primary sub-projects of LLNM are:	October 10-11, 2023 - LLVM Dev Mtg
LLVM Logo Blog	1. The LLVM Core libraries provide a modern source- and target-independent optimizer, along with code generation support for many popular CPUs (as well as some less common onest) These libraries are built around a well-specified code representation known as the LLVM intermediate representation ('LLVM IR'). The LLVM Core libraries are well documented, and it is particularly easy to invent your own language (or port an existing compiler) to use LLVM as an optimizer	ACM Software System Award!
Meetings LLVM Foundation	and code generator. 2. Clang is an 'LLVM native' C/C++/Objective-C compiler, which aims to deliver amazingly fast compiles, extremely useful <u>error and warning messages</u> and to provide a platform for building great source level tools. The Clang <u>Static Analyzer</u>	LLVM has been awarded the 2012 ACM Software System Award! This award is given by ACM
Download!	and clangeddy, are tools that automatically find bags in your code, and are great examples of the sort of tools that can be built using the Clang frontend as a library to parse CIC++ code.	every year. LLVM is in highly distinguished company! Click on any of the individual recipients' n citation describing the award.
Download now: <u>LLVM 16.0.6</u> All Releases	3. The LLDB project builds on libraries provided by LLVM and Clang to provide a great native debugger. It uses the Clang ASTs and expression parser, LLVM JIT, LLVM disastembler, etc so that it provides an experience that "just works". It is also blazing fast and much more memory efficient than GDB at loading symbols.	Upcoming Releases
APT Packages Fedora Snapshot	4. The liber+2 and liber+AHI projects provide a standard conformant and high-performance implementation of the C++ Standard Library, including full support for C++11 and C++14.	LLVM Release Schedule:
Packages Pre-releases	5. The compiler of project provides highly tuned implementations of the low-level code generator support routines like *_tizensdidi1* and other calls generated when a target doesn't have a short sequence of native instructions to implement a core IR operation. It also provides implementations of run-time libraries for dynamic testing tools such as <u>AddressSanitizer</u> , <u>MemorySanitizer</u> , <u>MemorySanitar</u> , <u>MemorySanitar</u> , <u>MemorySanitizer</u> , <u>MemoryS</u>	 17.0.x Jul 28th: 17.0.0-rc1 was released
View the open-source license	6. The MLIR subproject is a novel approach to building reusable and extensible compiler infrastructure. MLIR aims to address software fragmentation, improve compilation for heterogeneous hardware, significantly reduce the cost of building domain specific compilers, and aid in connecting existing compilers together.	 Aug 8th: 17.0.0-rc2 was released Aug 22nd: 17.0.0-rc3 was released Sep 5th: 17.0.0-rc4 was released
Search this Site	7. The <u>OpenMP</u> subproject provides an <u>OpenMP</u> runtime for use with the OpenMP implementation in Clang.	 Sep 19th: 17.0.1 was released Oct 3rd: 17.0.2 was released
Searchi	8. The pathy project implements a suite of cache-locality optimizations as well as auto-parallelism and vectorization using a polyhedral model.	 Oct 17th: 17.0.3 Oct 31th: 17.0.4
Useful Links	9. The libede project aims to implement the OpenCL standard library.	 Nov 14th: 17.0.5
Forums LLVM Discourse	10. The kite project implements a "symbolic virtual machine" which uses a theorem prover to try to evaluate all dynamic paths through a program in an effort to find bugs and to prove properties of functions. A major feature of kiee is that it can produce a testcase in the event that it detects a bug.	Developer Meetings
Mailing Lists: Commits List	11. The <u>LLD</u> project is a new linker. That is a drop-in replacement for system linkers and runs much faster.	Upcoming:
Discord (Real-time Chat):	12. The ROLT project is a post-link optimizer. It achieves the improvements by optimizing application's code layout based on execution profile gathered by sampling profiler.	 October 10-12, 2023
Discord	In addition to official subprojects of LLVM, there are a broad variety of other projects that <u>use components of LLVM is for various tasks</u> . Through these external projects you can use LLVM to compile Rayle, Python, Haskell, Rast, D, FHP. Pare, Las, Dalia, and a number of other languages. A major strength of LLVM is its version in the strength off affect tasks: every finite from doing labor weight. To compile a classical and and the strength off affect tasks: every finite tasks: e	Proceedings from past meetings: • May 10-11.2023
IRC Channel: irc.oftc.net #llvm	Ingrages like Lus to compiling Fortran code for massive super computers. As much as everything else, LLVM has a bread and friendly community of people who are interested in building great low-level tools. If you are interested in <u>getting involved</u> , a good first place is to skim the <u>LLVM Blog</u> and join <u>LLVM Discourse</u> .	November 8-9, 2022 May 10-11, 2022
Calendar: LLVM Community	Per index at every image energy in a grant per constraint of the period	November 16-19, 2021 October 6-8, 2020
Calendar		October 22-23.2019 April 8-9.2019
Dev. Resources: doxygen		October 17-18, 2018 April 16-17, 2018
Sources (GitHub) Code Review		October 18-19, 2017 March 27-28, 2017
Blog Bug tracker		November 3-4, 2016 March 17-18, 2016
Buildbot Green Dragon		October 29-30, 2015 April 13-14, 2015
LNT Scan-build		October 28-29, 2014 April 7-8, 2014
llvm-cov	LLVM Coverage bot	 April 7-8, 2014 Nov 6-7, 2013

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)
11db/	70.43% (18308/25995)	59.53% (185200/311089)	58.65% (118152/201447)	47.19% (62080/131558)
11vm/	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 Ilvm-cov -- Ilvm 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