

Making Linux Fly: Towards a Certified Linux Kernel

Wentao Zhang

University of Illinois at Urbana-Champaign

<wentaoz5@illinois.edu>



Steven H. VanderLeest

The Boeing Company

<Steven.H.VanderLeest@boeing.com>



Abstract

Although the Linux operating system has been used widely in many industries, adoption in aerospace has been slow due to the rigorous assurance evidence required as part of flight certification. The guidance for commercial flight software in most of the world is RTCA DO-178C, which identifies five progressively more rigorous levels of assurance. Providing the software life cycle data outlined by DO-178C is a daunting task for software as large and complex as Linux. In this project we focus on three objectives from DO-178C related to code coverage -- the fraction of the source code that is exercised by testing. The three types of code coverage in DO-178C are statement coverage, decision coverage, and Modified Condition/Decision Coverage (MC/DC). The last of these, MC/DC, is only required for Software Level A, the highest level of assurance.

For operating system kernels like Linux, measuring code coverage is challenging because of the unique execution environment compared to user space. Measuring MC/DC is even harder given the intricacy of the metric and limitations of tools. We share our experience in measuring Linux kernel's code coverage, with an emphasis on MC/DC. We describe how we have enabled measuring Linux kernel's MC/DC for the first time, by enhancing both the toolchain and the kernel itself. We also discuss the generalizability of our approach across different kernel versions and opportunities for improving coverage with kernel testing suites like KUnit and kselftest.

Agenda

- Introduction
- Software and Flight Certification
 - DO-178C guidance
 - Test coverage types
 - Motivation of work
- Filling the gaps in both LLVM-Cov and Linux kernel
 - Overview of contributions
 - First ever measurements of MC/DC coverage of the Linux kernel
 - Examples
 - Details on enhancements
 - Infrastructure and toolchain for measuring MC/DC of the Linux kernel
 - Demonstration
- Future Work

Introduction

- Who are we?
- Why are we working on this?
- What do we hope to achieve?
- Why should you be interested?

DO-178C Guidance for Flight Software

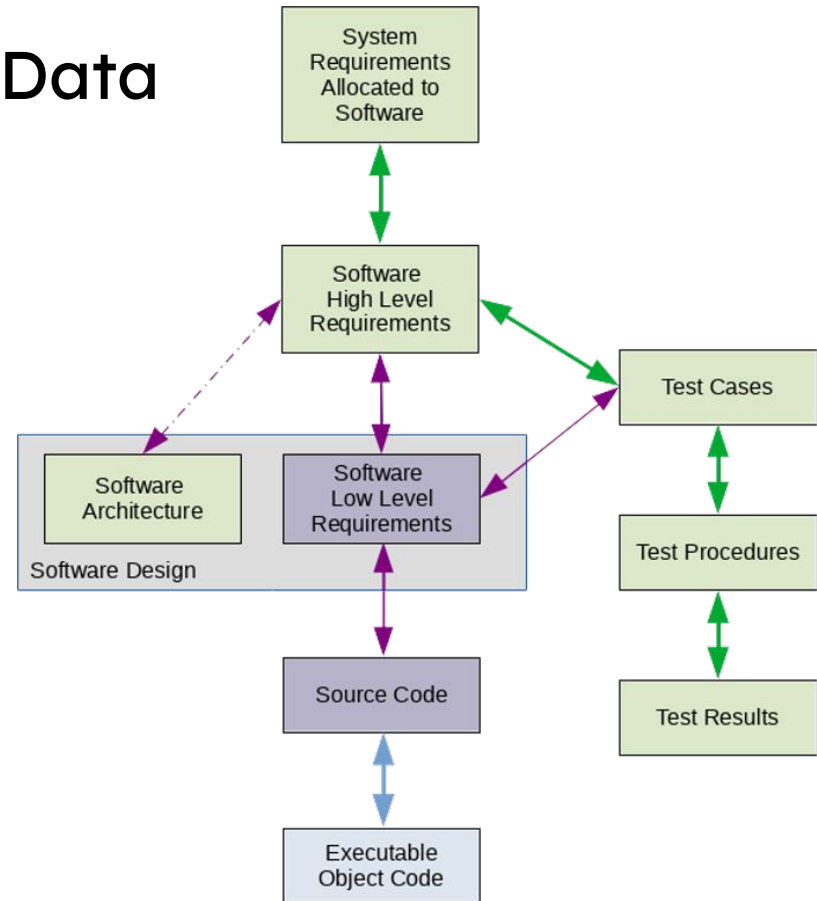
RTCA DO-178C/ED12-C is the primary guidance for developing safety-critical software for aircraft. It outlines 5 Software Levels with increasing impact and thus increasing rigor of assurance.

Level	Failure condition	Objectives
A	Catastrophic	71
B	Hazardous	69
C	Major	62
D	Minor	26
E	No Safety Effect	0

DO-178C Software Lifecycle Data

Evidence of software safety and reliability is produced when following the DO-178C processes and meeting the objectives. Many of the artifacts are linked in a traceability matrix.

Test coverage is the fraction of the Code exercised by the Test Cases/Procedures (normally should be 100%).



Steven H. VanderLeest, CC BY-SA 3.0
<<https://creativecommons.org/licenses/by-sa/3.0/>>, via Wikimedia Commons

Test Coverage

DO-178C Table A-7 “Verification of Verification Process Results”

ID	Test Coverage of Software Structure	Software Level	Full Coverage if
5	Modified condition/decision coverage	A	Each condition in each decision takes all values and shown to independently affect outcome
6	Decision coverage	B	Each decision takes all values
7	Statement coverage	C	Each statement executed

Test Coverage

Software Under Test

```
int fruit(int apples, int pears) {  
    int sum = 0;  
    if (apples > 0 && pears > 0) {  
        sum = apples + pears;  
    }  
    return sum;  
}
```

Test Program

```
fruit(3, 5);
```

Achieves full statement coverage

Test Coverage

Software Under Test

```
int fruit(int apples, int pears) {  
    int sum = 0;  
    if (apples > 0 && pears > 0) {  
        sum = apples + pears;  
    }  
    return sum;  
}
```

Test Program

```
fruit(3, 5);  
fruit(0, 0);
```

Achieves full statement and
decision coverage

Test Coverage

Software Under Test

```
int fruit(int apples, int pears) {  
    int sum = 0;  
    if (apples > 0 && pears > 0) {  
        sum = apples + pears;  
    }  
    return sum;  
}
```

Test Program

```
fruit(1, 2);  
fruit(0, 2);  
fruit(1, 0);
```

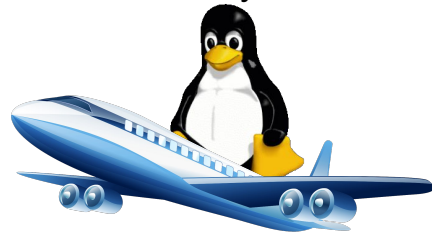
Achieves full statement, decision,
and MC/DC coverage

Motivation of Work

- At least two open source coverage tools available that can measure coverage inside the Linux kernel: gcov and LLVM-cov.
- Boeing and UIUC are collaborating to enhance both LLVM-cov and Linux kernel to achieve all DO-178C coverage objectives

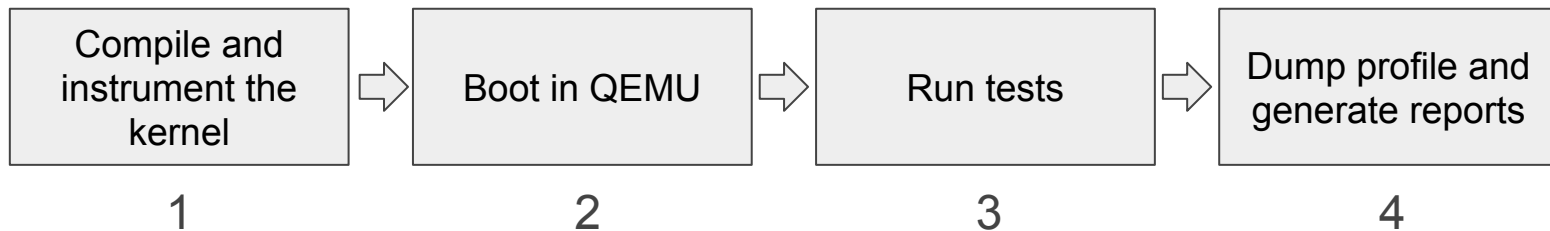
Certifying Linux to DO-178C

- Goal: achieving all DO-178C objectives 5 through 9 for the Linux kernel
 - Objective 5: Test coverage of structure (**modified condition/decision**) is achieved
 - Objective 6: Test coverage of structure (decision coverage) is achieved
 - Objective 7: Test coverage of structure (statement coverage) is achieved
 - Objective 9: Verification of additional code, that cannot be traced to source code, is achieved
- Our current focus: **enabling measurement** of **MC/DC** for Linux
 - We are not aiming at achieving high coverage (yet)
 - Instead, we provide the tools to report MC/DC and ensure the tools' reliability



Overview of Contributions

- We have built the *first* infrastructure for measuring Linux kernel's MC/DC
 - Integrating **open-source** solutions
- Key components
 - **Tool:** **Clang/LLVM** version ≥ 18
 - **Target:** Linux kernel v5.15.127
 - **Tests:** **KUnit**, **kseltest**, LTP, syzkaller...
- Steps:



Overall Coverage Report Including MC/DC

Coverage Report

Created: 2024-02-15 19:51

Click [here](#) for information about interpreting this report.

[Go to the summary end](#)

Filename	Function Coverage	Line Coverage	Branch Coverage	MC/DC
linux/arch/x86/entry/common.c	20.00% (1/5)	22.86% (8/35)	12.50% (1/8)	0.00% (0/1)
linux/arch/x86/entry/vsyscall/vsyscall	27.27% (3/11)	10.00% (20/200)	6.25% (6/96)	7.69% (1/13)
linux/arch/x86/events/amd/core.c	0.00% (0/28)	0.00% (0/352)	0.00% (0/268)	0.00% (0/18)
linux/arch/x86/events/amd/ibs.c	5.26% (2/38)	2.85% (16/561)	1.85% (5/270)	0.00% (0/22)
⋮				
linux/sound/pci/hda/hda_sysfs.c	0.00% (0/9)	0.00% (0/45)	0.00% (0/4)	- (0/0)
linux/sound/sound_core.c	60.00% (3/5)	58.33% (14/24)	50.00% (3/6)	- (0/0)
Totals	19.65% (16877/85908)	12.52% (172414/1376595)	8.31% (54418/655242)	2.20% (1400/63772)

[Summary page](#) for the whole kernel source tree

Simplest Example: 2-Condition Decision ([kernel/softirq.c](#))

458

7

```
if (pending && !ksoftirqd_running(pending))
```

MC/DC Decision Region (458:6) to (458:44)

Number of Conditions: 2

Condition C1 --> (458:6)

Condition C2 --> (458:17)

Executed MC/DC Test Vectors:

	C1, C2	Result
1	{ T, F }	= F }
2	{ T, T }	= T }

C1-Pair: not covered

C2-Pair: covered: (1,2)

MC/DC Coverage for Expression: 50.00%

Example of a 6-Condition Decision ([drivers/pci/quirks.c](#))

```
63 8.91k if ((f->class == (u32) (dev->class >> f->class_shift) ||  
64 8.91k     f->class == (u32) PCI_ANY_ID) &&  
65 8.91k     (f->vendor == dev->vendor ||  
66 7.42k     f->vendor == (u16) PCI_ANY_ID) &&  
67 8.91k     (f->device == dev->device ||  
68 1.54k     f->device == (u16) PCI_ANY_ID)) {
```

MC/DC Decision Region (63:7) to (68:8)

Number of Conditions: 6
Condition C1 --> (63:8)
Condition C2 --> (64:8)
Condition C3 --> (65:8)
Condition C4 --> (66:8)
Condition C5 --> (67:8)
Condition C6 --> (68:8)

Executed MC/DC Test Vectors:

	C1, C2, C3, C4, C5, C6	Result
1	{ F, F, -, -, -, - }	= F }
2	{ T, -, F, F, -, - }	= F }
3	{ F, T, F, F, -, - }	= F }
4	{ F, T, T, -, F, F }	= F }
5	{ F, T, T, -, T, - }	= T }
6	{ F, T, T, -, F, T }	= T }
7	{ T, -, F, T, F, T }	= T }

C1-Pair: covered: (1,7)
C2-Pair: covered: (1,5)
C3-Pair: covered: (3,5)
C4-Pair: covered: (2,7)
C5-Pair: covered: (4,5)
C6-Pair: covered: (4,6)

MC/DC Coverage for Expression: 100.00%

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

```
$ clang -Xclang -dump-coverage-mapping
```

File 0, 1:23 -> 5:2 = #0

File 0, 2:9 -> 2:27 = #0

File 0, 2:9 -> 2:16 = #0

Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)

File 0, 2:20 -> 2:27 = #2

Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)

Gap,File 0, 2:28 -> 3:9 = #1

File 0, 3:9 -> 3:17 = #1

Gap,File 0, 3:18 -> 4:5 = (#0 - #1)

File 0, 4:5 -> 4:13 = (#0 - #1)

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

```
$ clang -Xclang -dump-coverage-mapping
```

```
File 0, 1:23 -> 5:2 = #0
```

```
File 0, 2:9 -> 2:27 = #0
```

```
File 0, 2:9 -> 2:16 = #0
```

```
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)
```

```
File 0, 2:20 -> 2:27 = #2
```

```
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)
```

```
Gap,File 0, 2:28 -> 3:9 = #1
```

```
File 0, 3:9 -> 3:17 = #1
```

```
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)
```

```
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Code region: How many times the curly brace is entered?

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

```
$ clang -Xclang -dump-coverage-mapping
```

File 0, 1:23 -> 5:2 = #0

File 0, 2:9 -> 2:27 = #0

File 0, 2:9 -> 2:16 = #0

Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)

File 0, 2:20 -> 2:27 = #2

Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)

Gap,File 0, 2:28 -> 3:9 = #1

File 0, 3:9 -> 3:17 = #1

Gap,File 0, 3:18 -> 4:5 = (#0 - #1)

File 0, 4:5 -> 4:13 = (#0 - #1)

Code region: How many times the decision is evaluated?

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

```
$ clang -Xclang -dump-coverage-mapping
```

File 0, 1:23 -> 5:2 = #0

File 0, 2:9 -> 2:27 = #0

File 0, 2:9 -> 2:16 = #0

Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)

File 0, 2:20 -> 2:27 = #2

Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)

Gap,File 0, 2:28 -> 3:9 = #1

File 0, 3:9 -> 3:17 = #1

Gap,File 0, 3:18 -> 4:5 = (#0 - #1)

File 0, 4:5 -> 4:13 = (#0 - #1)

Code region: How many times the 1st condition is evaluated?

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

```
$ clang -Xclang -dump-coverage-mapping
```

File 0, 1:23 -> 5:2 = #0

File 0, 2:9 -> 2:27 = #0

File 0, 2:9 -> 2:16 = #0

Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)

File 0, 2:20 -> 2:27 = #2

Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)

Gap,File 0, 2:28 -> 3:9 = #1

File 0, 3:9 -> 3:17 = #1

Gap,File 0, 3:18 -> 4:5 = (#0 - #1)

File 0, 4:5 -> 4:13 = (#0 - #1)

Code region: How many times the 2nd condition is evaluated?

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

The same location is also a branch region

```
$ clang -Xclang -dump-coverage-mapping
```

File 0, 1:23 -> 5:2 = #0

File 0, 2:9 -> 2:27 = #0

File 0, 2:9 -> 2:16 = #0

Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)

File 0, 2:20 -> 2:27 = #2

Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)

Gap,File 0, 2:28 -> 3:9 = #1

File 0, 3:9 -> 3:17 = #1

Gap,File 0, 3:18 -> 4:5 = (#0 - #1)

File 0, 4:5 -> 4:13 = (#0 - #1)

Branch region: How many times the 1st condition is evaluated to "true" and "false" respectively?

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

The same location is also a branch region

```
$ clang -Xclang -dump-coverage-mapping
```

```
File 0, 1:23 -> 5:2 = #0  
File 0, 2:9 -> 2:27 = #0  
File 0, 2:9 -> 2:16 = #0  
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)  
File 0, 2:20 -> 2:27 = #2  
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)  
Gap,File 0, 2:28 -> 3:9 = #1  
File 0, 3:9 -> 3:17 = #1  
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)  
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Branch region: How many times the 2st condition is evaluated to "true" and "false" respectively?

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

```
$ clang -Xclang -dump-coverage-mapping
```

File 0, 1:23 -> 5:2 = #0

File 0, 2:9 -> 2:27 = #0

File 0, 2:9 -> 2:16 = #0

Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)

File 0, 2:20 -> 2:27 = #2

Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)

Gap,File 0, 2:28 -> 3:9 = #1

File 0, 3:9 -> 3:17 = #1

Gap,File 0, 3:18 -> 4:5 = (#0 - #1)

File 0, 4:5 -> 4:13 = (#0 - #1)

Code region: How many times the 1st return statement is executed?

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

```
$ clang -Xclang -dump-coverage-mapping
```

File 0, 1:23 -> 5:2 = #0

File 0, 2:9 -> 2:27 = #0

File 0, 2:9 -> 2:16 = #0

Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)

File 0, 2:20 -> 2:27 = #2

Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)

Gap,File 0, 2:28 -> 3:9 = #1

File 0, 3:9 -> 3:17 = #1

Gap,File 0, 3:18 -> 4:5 = (#0 - #1)

File 0, 4:5 -> 4:13 = (#0 - #1)

Code region: How many times the final return statement is executed?

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

```
int foo(int x, int y) {  
    if ((x > 0) && (y > 0))  
        return 1;  
    return 0;  
}
```

```
$ clang -Xclang -dump-coverage-mapping
```

```
File 0, 1:23 -> 5:2 = #0  
File 0, 2:9 -> 2:27 = #0  
File 0, 2:9 -> 2:16 = #0  
Branch,File 0, 2:9 -> 2:16 = #2, (#0 - #2)  
File 0, 2:20 -> 2:27 = #2  
Branch,File 0, 2:20 -> 2:27 = #3, (#2 - #3)  
Gap,File 0, 2:28 -> 3:9 = #1  
File 0, 3:9 -> 3:17 = #1  
Gap,File 0, 3:18 -> 4:5 = (#0 - #1)  
File 0, 4:5 -> 4:13 = (#0 - #1)
```

Four independent counters are maintained. Others can be derived from these four.

Illustration of LLVM-Cov Instrumentation

- Coverage mapping **regions** and **counters** are assigned

LLVM IR

```
%pgocount = load i64, ptr @__profc_foo(int, int), align 8
%0 = add i64 %pgocount, 1
store i64 %0, ptr @__profc_foo(int, int), align 8
```

x86-64

```
mov    rax, qword ptr [rip + .L__profc_foo(int, int)]
add    rax, 1
mov    qword ptr [rip + .L__profc_foo(int, int)], rax
```

`$ clang -Xclang -dump-coverage-mapping`

#0

#2

#3

#1

Counters under the hood: INC instruction and memory read/write

Infrastructure for measuring Linux MC/DC coverage

- **Tool:** Clang/LLVM version ≥ 18
 - We helped test the tool as early adopters and fixed/reported bugs
- **Target:** Linux kernel v5.15.127
 - We built the necessary kernel support to export the coverage profile
- **Tests:** KUnit and kselftest for now
 - Results shown are for KUnit




Toolchain

- MC/DC feature for Clang/LLVM was implemented and merged into mainline January 2024
 - Built on top of [Source-based Code Coverage](#)
 - Mostly contributed by Alan Phipps from Texas Instruments
 - Utilizing bitmaps to track test vectors
- Included in releases $\geq 18.1.0$ since March 2024
- We are actively testing and verifying Clang/LLVM MC/DC
 - We are among the first to test this implementation (and our target is very unique!)
 - Collaborating with the upstream, we fixed/reported a few bugs (see next slides)






Our Contributions to LLVM

Reported

-  [\[clang\]\[CoverageMapping\] "Assertion AfterLoc.isValid\(\) failed" during compiling switch within statement expressions](#) [llvm/llvm-project#86998](#)
-  [\[llvm-cov\]\[MC/DC\] "Branch not found in Decisions" when handling complicated macros](#) [llvm/llvm-project#87000](#)
-  [\[llvm-cov\]\[nit\] inconsistent variable name of type MCDCView](#) [llvm/llvm-project#80935](#)

Merged

-  [\[llvm-cov\]\[CoverageView\] minor fix/improvement to HTML and text coverage output](#) [llvm/llvm-project#80952](#)
-  [\[clang\]\[CodeGen\] Keep processing the rest of AST after encountering unsupported MC/DC expressions](#) [llvm/llvm-project#82464](#)
-  [\[clang\]\[CoverageMapping\] do not emit gap when either end is an ImplicitValueInitExpr](#) [llvm/llvm-project#89564](#)
- As well as back porting them to `release/18.x`
 -  [Backport 0bf4f82 to release/18.x](#) [llvm/llvm-project#82571](#)
 -  [release/18.x: \[clang\]\[CodeGen\] Keep processing the rest of AST after encountering unsupported MC/DC expressions \(#82464\)](#) [llvm/llvm-project#82866](#)

Pending

-  [\[Coverage\]\[Expansion\] handle nested macros in scratch space](#) [llvm/llvm-project#89869](#)
-  [\[Coverage\]\[docs\] for the main region array there shouldn't be a numRegionArrays : LEB128 field](#) [llvm/llvm-project#80531](#)
-  [\[llvm-cov\] Keep the detailed error message in CoverageMappingIterator](#) [llvm/llvm-project#80415](#)

Superseded

-  [\[llvm-cov\] assertion failure when handling MC/DC that involves macros](#) [llvm/llvm-project#80098](#)

Our Contributions to LLVM

- Highlights:

- [#86998](#) (and fix in [#89564](#)): uncovered by instrumenting `fs/coredump.c`, which uses a non-standard C syntax
- [#87000](#) (and fix in [#89869](#)): two independent problems, both uncovered by instrumenting `drivers/iommu/intel/perfmon.c`, involving complicated macros

```
struct Foo foo = {  
    .field1 = ({  
        switch (123) {  
            case 123:  
                break;  
        }  
        456;  
    } ),  
};
```

Reduced #86998

```
#define FOO(x) foo_##x  
int a, foo_b;  
if (a && FOO(b)) { ... }
```

Reduced #87000

- Even with these fixes, Clang/LLVM coverage does **not** work out-of-the-box to measure code coverage of the Linux kernel

Kernel Support for Linux MC/DC

- In-memory counters and bitmaps need to be exported
 - User space would be straightforward: write to a file at the same directory as the executable
 - For the kernel image: no concept of “parent directory” – write to **a pseudo file system** instead
 - It is also the practice of how gcov exports its kernel profile
 - For LLVM Source-based Code Coverage, however, not yet supported
- Reuse some code of an early Linux kernel [patch](#) from 2021
 - It was meant for performance optimization instead of coverage itself
 - Over the years, profile data format has also changed in LLVM ([from version 5 to version 9](#))
 - Prepare a new patch by (1) focusing on *coverage* functionality instead of *profiling* and (2) synchronizing the data format
- We are preparing to upstream our patch to the Linux kernel

Generalizability of Our Kernel Patch

- To further validate our infrastructure, and expose potential bugs: apply our infra to other kernel versions
- Target versions: 9 branches listed in <https://www.kernel.org/> in March
- A small kernel config from our university's OS class

Version	Results (using UIUC CS 423 config)
mainline 6.9-rc1	Works fine
stable 6.8.2	Warning at visualization stage
stable 6.7.11	Warning at visualization stage
longterm 6.6.23 longterm 6.1.83 longterm 5.15.153 longterm 5.10.214 longterm 5.4.273 longterm 4.19.311	Works fine <ul style="list-style-type: none">• No crash or llmv-cov warning• The end reports can be produced

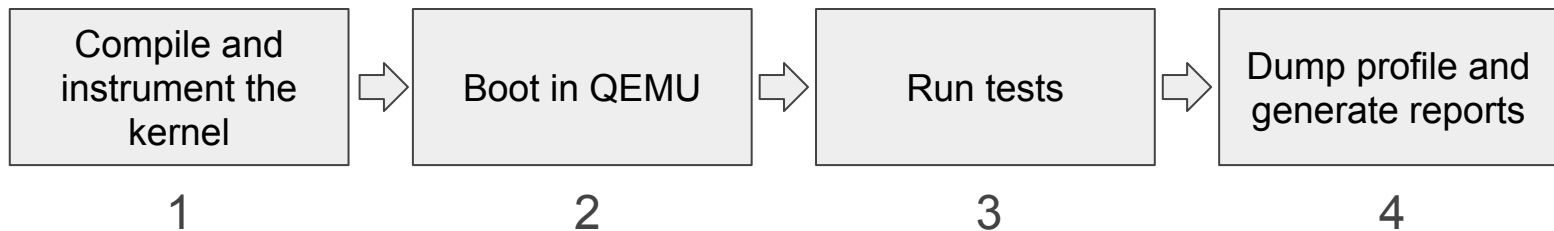
Exercise Various Kernel Testing Techniques

- Coverage report with different tests:

	Function Coverage	Line Coverage	MC/DC
Boot	19.45%	12.29%	2.08%
Boot + KUnit	19.66% ↑0.21%	12.45% ↑0.16%	2.13% ↑0.05%
Boot + KUnit + kselftest	25.99% ↑6.33%	16.87% ↑4.42%	3.44% ↑1.31%

- Next: Try additional test suites such as LTP, KCOV/syzkaller...

Demo



Summary and Future Work

- We can measure **MC/DC** of **Linux kernel**
 - Tested with kernel v5.15 (and more) and Clang/LLVM 18
 - Will be open sourced in <https://github.com/xlab-uiuc/linux-mcdc> as soon as we get the clearance

Next steps:

- Test the tools more thoroughly to find potential bugs and improve the tools
 - Check accuracy and fidelity of the current implementation
 - Improve the presentation of data in the report
 - Compare with proprietary tools like VectorCAST
 - Collaborate with the Clang/LLVM upstream
- Promote the kernel support to Linux upstream
- Data coupling and control coupling coverage (DO-178C objective 8)
- Object coverage (DO-178C objective 9)
- DO-330 Tool Qualification

Acknowledgement

- Open-source community
 - LLVM developers: Alan Phipps, @chapuni, @ZequanWu, @ornata, @hanickadot, @MaskRay...
 - GCC developers: Jørgen Kvalsvik...
 - Kernel developers: Sami Tolvanen, Bill Wendling...
- UIUC work is supported with funding from The Boeing Company