Source-based code coverage of Linux kernel

Wentao Zhang

Tingxu Ren, Jinghao Jia, Darko Marinov, Tianyin Xu

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Precise source-based coverage is important

Precise source-based coverage is important

- For testing: improve software quality

Precise source-based coverage is important

- For testing: improve software quality
  - Coverage can better reflect developer intent

Precise source-based coverage is important

- For testing: improve software quality
  - Coverage can better reflect developer intent
- For certification

Precise source-based coverage is important

- For testing: improve software quality
  - Coverage can better reflect developer intent
- For certification
  - Aviation: DO-178C
  - Rail: EN 50128
  - Automotive: ISO 26262

Precise source-based coverage is important

- For testing: improve software quality
  - Coverage can better reflect developer intent

Related (past) talk:
“Measuring and Understanding Linux Kernel Tests”
Kernel Testing & Dependability MC

- For certification
  - Aviation: DO-178C
  - Rail: EN 50128
  - Automotive: ISO 26262

Related (past) talk:
“Making Linux Fly: Towards a Certified Linux Kernel”
Refereed track

Existing coverage tools in kernel

Existing coverage tools in kernel

- `KCOV`

- `gcov`

Existing coverage tools in kernel

- `KCOV`
  - Based on `-fsanitize-coverage` compiler flag
- `gcov`

Existing coverage tools in kernel

- `KCOV`
  - Based on `-fsanitize-coverage` compiler flag
  - Designed for fuzzing, only having rudimentary reports
- `gcov`

Existing coverage tools in kernel

- `KCOV`
  - Based on `-fsanitize-coverage` compiler flag
  - Designed for fuzzing, only having rudimentary reports
  - Relying on **debuginfo** etc. to map back to source code
- `gcov`

Existing coverage tools in kernel

- `KCOV`
  - Based on `-fsanitize-coverage` compiler flag
  - Designed for fuzzing, only having rudimentary reports
  - Relying on **debuginfo** etc. to map back to source code
- `gcov`
  - Based on `-fprofile-arcs -ftest-coverage` compiler flags

Existing coverage tools in kernel

- `KCOV`
  - Based on `-fsanitize-coverage` compiler flag
  - Designed for fuzzing, only having rudimentary reports
  - Relying on **debuginfo** etc. to map back to source code
- `gcov`
  - Based on `-fprofile-arcs -ftest-coverage` compiler flags
  - Designed for general coverage

Existing coverage tools in kernel

- `KCOV`
  - Based on `-fsanitize-coverage` compiler flag
  - Designed for fuzzing, only having rudimentary reports
  - Relying on **debuginfo** etc. to map back to source code
- `gcov`
  - Based on `-fprofile-arcs -ftest-coverage` compiler flags
  - Designed for general coverage
  - Relying on **`*.gcno` files** to map back to source code

Existing coverage tools in kernel

- `KCOV`
  - Based on `-fsanitize-coverage` compiler flag
  - Designed for fuzzing, only having rudimentary reports
  - Relying on **debuginfo** etc. to map back to source code
- `gcov`
  - Based on `-fprofile-arcs -ftest-coverage` compiler flags
  - Designed for general coverage
  - Relying on **`*.gcno` files** to map back to source code
  - But reports can still be confusing (examples in later slides)

Existing coverage tools in kernel

- KCOV
  - Based on `-fsanitize-coverage` compiler flag
  - Designed for fuzzing, only having rudimentary reports
  - Relying on **debuginfo** etc. to map back to source code
- gcov
  - Based on `-fprofile-arcs -ftest-coverage` compiler flags
  - Designed for general coverage
  - Relying on **`*.gcno` files** to map back to source code
  - But reports can still be confusing (examples in later slides)

Both can only be **approximately** correlated to source code.
Both are **susceptible to optimization**.

Motivating example: idealized world

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
static struct drm_buddy_block *
__get_buddy(struct drm_buddy_block *block)
{
    struct drm_buddy_block *parent;

    parent = block->parent;
    if (!parent)
        return NULL;


    if (parent->left == block)
        return parent->right;


    return parent->left;
}
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

Line coverage

```
static struct drm_buddy_block *
__get_buddy(struct drm_buddy_block *block)
{
    struct drm_buddy_block *parent;

    parent = block->parent;
    if (!parent)
        return NULL;


    if (parent->left == block)
        return parent->right;


    return parent->left;
}
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
  Line      static struct drm_buddy_block *
coverage    __get_buddy(struct drm_buddy_block *block)
            {
                struct drm_buddy_block *parent;

       N1       parent = block->parent;
                if (!parent)
                    return NULL;


                if (parent->left == block)
                    return parent->right;


                return parent->left;
            }
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
  Line      static struct drm_buddy_block *
coverage    __get_buddy(struct drm_buddy_block *block)
            {
                struct drm_buddy_block *parent;

      N1        parent = block->parent;
      N1        if (!parent)
                    return NULL;


                if (parent->left == block)
                    return parent->right;


                return parent->left;
            }
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
    Line      static struct drm_buddy_block *
  coverage    __get_buddy(struct drm_buddy_block *block)
              {
                  struct drm_buddy_block *parent;

       N_1        parent = block->parent;
       N_1        if (!parent)
       N_2            return NULL;  /* then clause */


                  if (parent->left == block)
                      return parent->right;


                  return parent->left;
              }
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

Line coverage	
	`static struct drm_buddy_block *`
	`__get_buddy(struct drm_buddy_block *block)`
	`{`
	`    struct drm_buddy_block *parent;`
N_1	`    parent = block->parent;`
N_1	`    if (!parent)`
N_2	`        return NULL; /* then clause */`
	`    /* else clause */`
N_1-N_2	`    if (parent->left == block)`
	`        return parent->right;`
	`    return parent->left;`
	`}`

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
    Line        static struct drm_buddy_block *
  coverage      __get_buddy(struct drm_buddy_block *block)
                {
                    struct drm_buddy_block *parent;

        N1          parent = block->parent;
        N1          if (!parent)
        N2              return NULL;  /* then clause */

                    /* else clause */
     N1-N2          if (parent->left == block)
                        return parent->right;


                    return parent->left;
                }
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
   Line       static struct drm_buddy_block *
 coverage     __get_buddy(struct drm_buddy_block *block)
              {
                  struct drm_buddy_block *parent;

        N_1       parent = block->parent;
        N_1       if (!parent)
        N_2           return NULL;  /* then clause */

                  /* else clause */
    N_1-N_2       if (parent->left == block)
        N_3           return parent->right;


                  return parent->left;
              }
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
Line coverage

                static struct drm_buddy_block *
                __get_buddy(struct drm_buddy_block *block)
                {
                    struct drm_buddy_block *parent;

N_1                 parent = block->parent;
N_1                 if (!parent)
N_2                     return NULL;  /* then clause */

                    /* else clause */
N_1-N_2             if (parent->left == block)
N_3                     return parent->right;


N_1-N_2-N_3         return parent->left;
                }
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
   Line       static struct drm_buddy_block *
 coverage     __get_buddy(struct drm_buddy_block *block)
              {
                  struct drm_buddy_block *parent;

       N_1        parent = block->parent;
       N_1        if (!parent)
       N_2            return NULL;


   N_1-N_2        if (parent->left == block)
       N_3            return parent->right; /* then clause */

                  /* else clause */
N_1-N_2-N_3       return parent->left;
              }
```

`drivers/gpu/drm/drm_buddy.c`

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
     Line       static struct drm_buddy_block *
   coverage     __get_buddy(struct drm_buddy_block *block)
                {
                    struct drm_buddy_block *parent;

          $N_1$     parent = block->parent;
          $N_1$     if (!parent)
          $N_2$         return NULL;


    $N_1$-$N_2$     if (parent->left == block)
          $N_3$         return parent->right;


$N_1$-$N_2$-$N_3$   return parent->left;
                }
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
    Line           static struct drm_buddy_block *
  coverage         __get_buddy(struct drm_buddy_block *block)
                   {
                       struct drm_buddy_block *parent;

            $N_1$      parent = block->parent;
            $N_1$      if (!parent)
            $N_2$          return NULL;
                         “True”  outcome taken   ??    times
                         “False” outcome taken   ??    times
      $N_1$-$N_2$      if (parent->left == block)
            $N_3$          return parent->right;
                         “True”  outcome taken   ??    times
                         “False” outcome taken   ??    times
$N_1$-$N_2$-$N_3$      return parent->left;
                   }
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

```
    Line      static struct drm_buddy_block *
  coverage    __get_buddy(struct drm_buddy_block *block)
              {
                  struct drm_buddy_block *parent;

        N1        parent = block->parent;
        N1        if (!parent)
        N2            return NULL;
                     “True”  outcome taken   N2     times
                     “False” outcome taken  N1-N2   times
     N1-N2        if (parent->left == block)
        N3            return parent->right;
                     “True”  outcome taken   N3     times
                     “False” outcome taken N1-N2-N3 times
  N1-N2-N3        return parent->left;
              }
```

drivers/gpu/drm/drm_buddy.c

Motivating example: idealized world

- What should an intuitive coverage report look like?

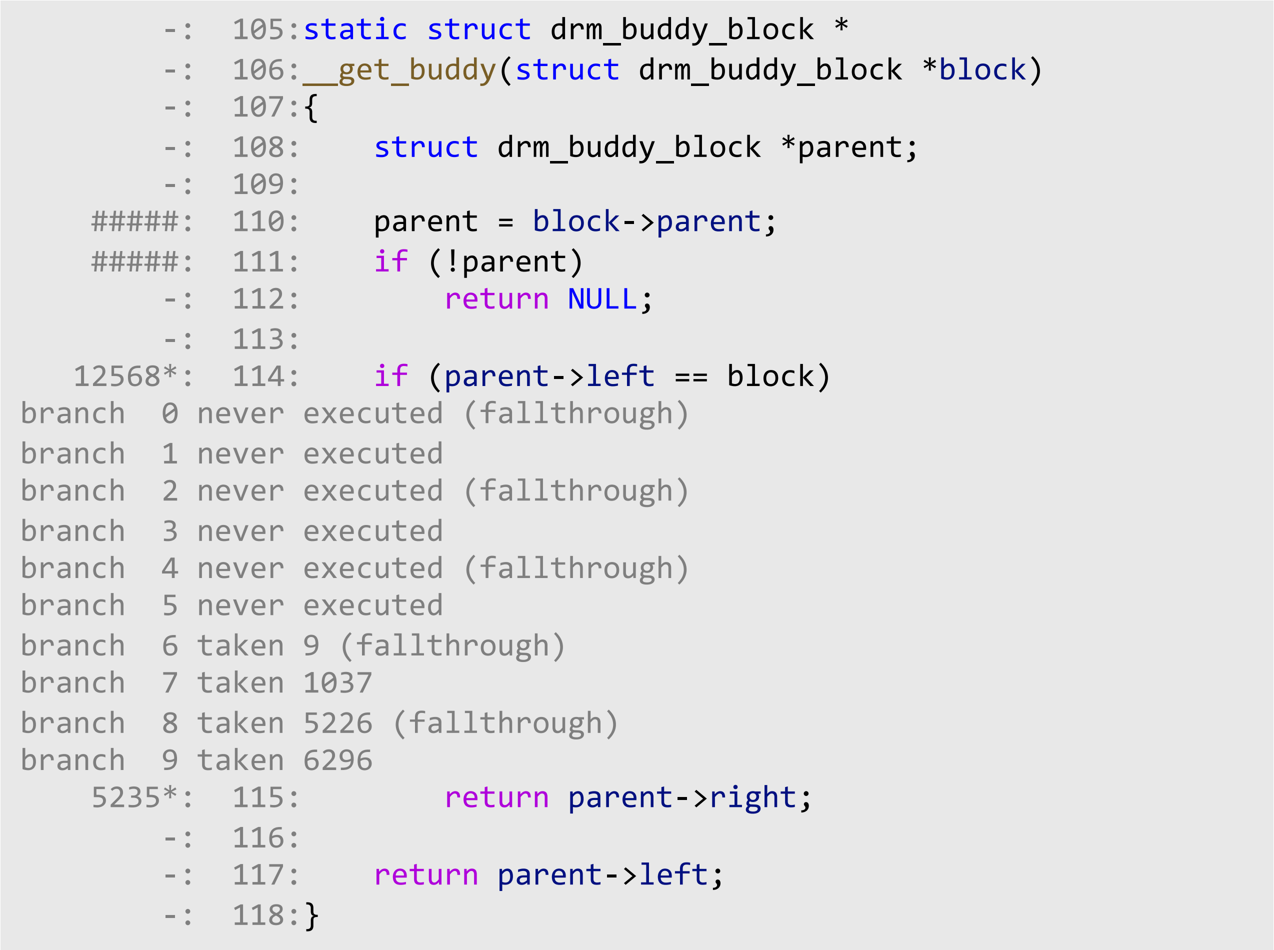
Line coverage	
	`static struct drm_buddy_block *`
	`__get_buddy(struct drm_buddy_block *block)`
	`{`
N_1-N_2	`    if (parent->left == block)`
N_3	`        return parent->right;`
	“True” outcome taken N_3 times “False” outcome taken $N_1-N_2-N_3$ times
$N_1-N_2-N_3$	`    return parent->left;`
	`}`

`drivers/gpu/drm/drm_buddy.c`

This is exactly our goal:
precise source correlation regardless of optimization level.

`gcov` reports in reality

gcov reports in reality



```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

`drivers/gpu/drm/drm_buddy.c`

- Setup
  - Kernel version: `v6.10-rc7`
  - Options
    - `defconfig`
    - `CONFIG_KUNIT_ALL_TESTS`
    - Default optimization level
  - Measurement span: kernel boot with all KUnit tests

gcov reports in reality

```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

drivers/gpu/drm/drm_buddy.c

gcov notation	Meaning
-	The line contains no code
#####	Unexecuted

gcov reports in reality

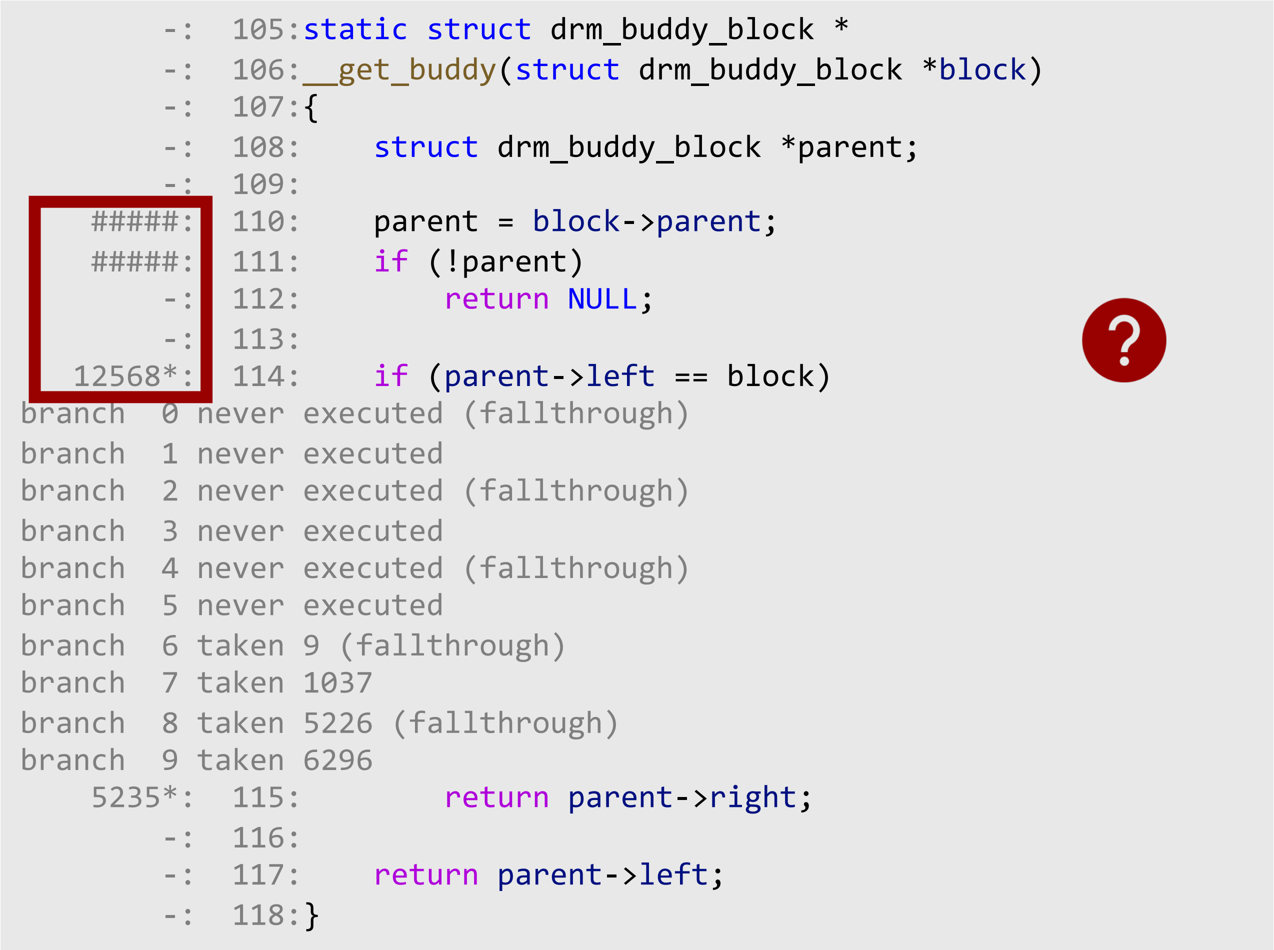
```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

?

drivers/gpu/drm/drm_buddy.c

gcov notation	Meaning
-	The line contains no code
#####	Unexecuted

gcov reports in reality



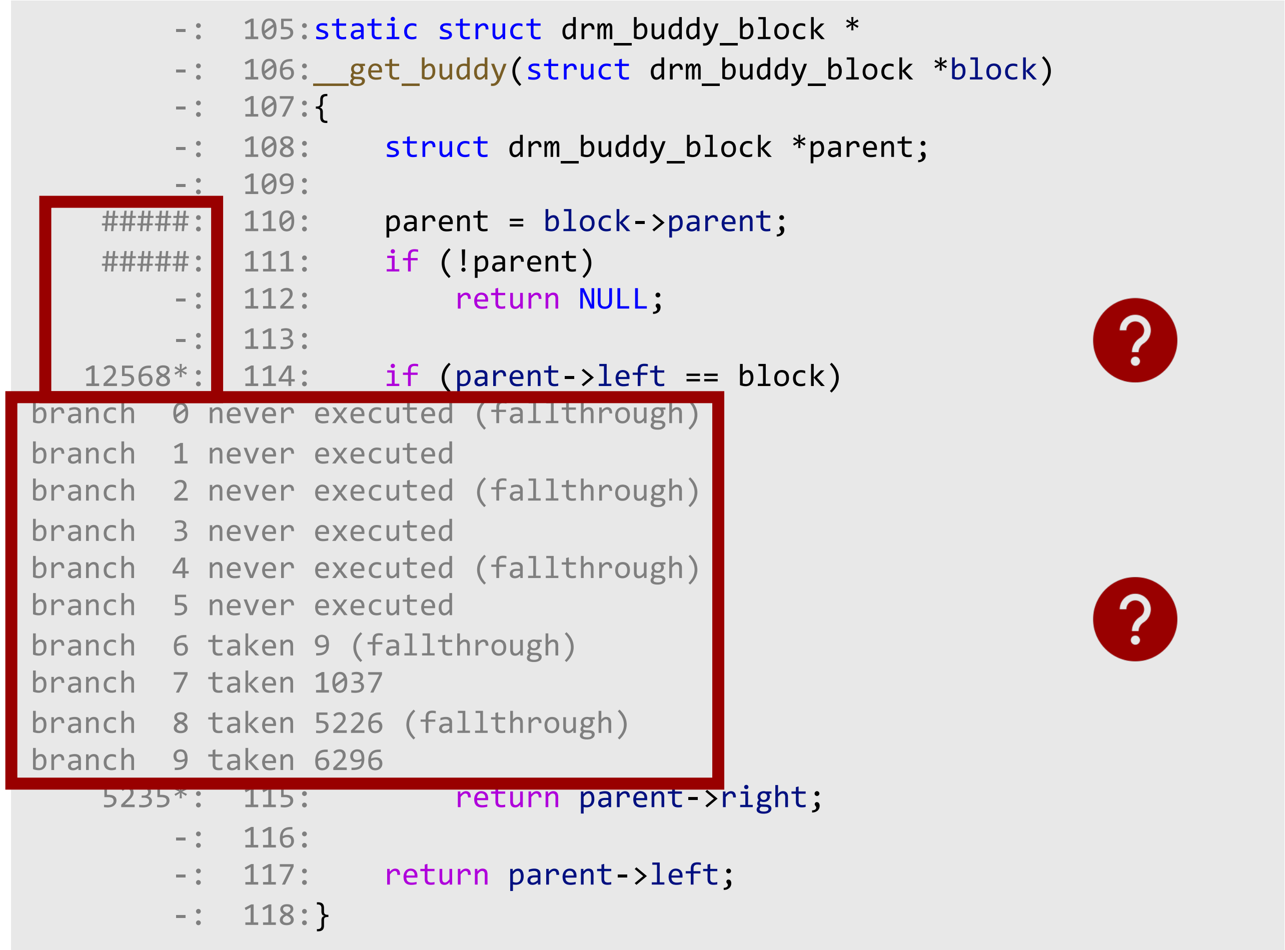
```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

drivers/gpu/drm/drm_buddy.c

gcov notation	Meaning
-	The line contains no code
#####	Unexecuted

- Line coverage:
  - Executed line (#L114) shows up after unexecuted line (#L110)

gcov reports in reality



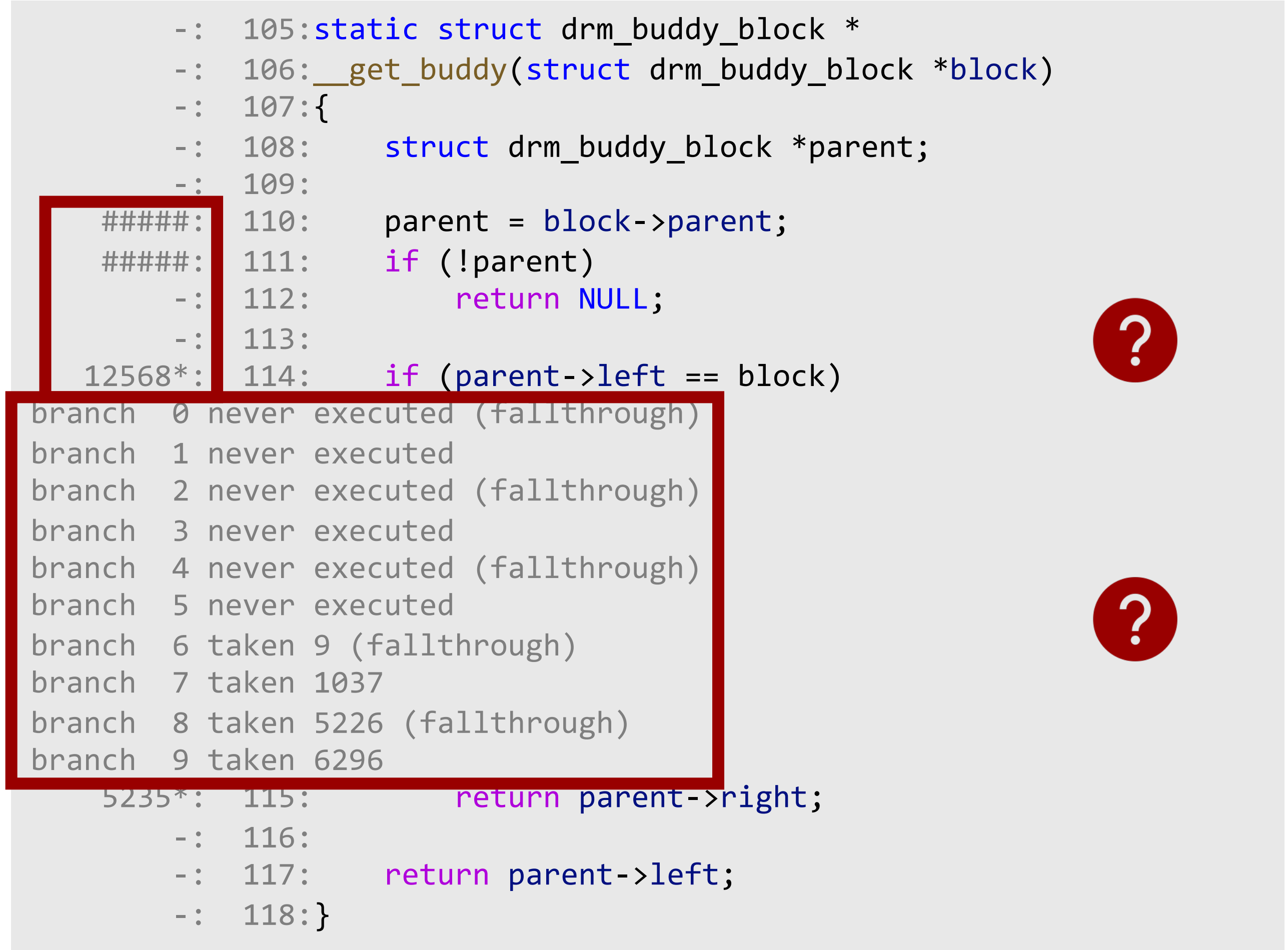
```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

drivers/gpu/drm/drm_buddy.c

gcov notation	Meaning
-	The line contains no code
#####	Unexecuted

- Line coverage:
  - Executed line (#L114) shows up after unexecuted line (#L110)

gcov reports in reality



```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

drivers/gpu/drm/drm_buddy.c

gcov notation	Meaning
-	The line contains no code
#####	Unexecuted

- Line coverage:
  - Executed line (#L114) shows up after unexecuted line (#L110)
- Branch coverage:
  - A simple `if` statement is reported to have 10 outcomes, instead of 2

`llvm-cov` and source-based code coverage

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter ⇔ source location**

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter ⇔ source location**
  - Location includes both line and column

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter** ⇔ **source location**
  - Location includes both line and column
- Instrument at frontend thus **not affected by optimization** [1]

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html#impact-of-llvm-optimizations-on-coverage-reports

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter ⇔ source location**
  - Location includes both line and column
- Instrument at frontend thus **not affected by optimization** [1]
  - Mapping is constructed at this stage and hence immutable

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html#impact-of-llvm-optimizations-on-coverage-reports

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter ⬄ source location**
  - Location includes both line and column
- Instrument at frontend thus **not affected by optimization** [1]
  - Mapping is constructed at this stage and hence immutable
  - Optimization passes preserve precise source-based coverage

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html#impact-of-llvm-optimizations-on-coverage-reports

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter ⇔ source location**
  - Location includes both line and column
- Instrument at frontend thus **not affected by optimization** [1]
  - Mapping is constructed at this stage and hence immutable
  - Optimization passes preserve precise source-based coverage
- User space usage

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html#impact-of-llvm-optimizations-on-coverage-reports

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter ⇔ source location**
  - Location includes both line and column
- Instrument at frontend thus **not affected by optimization** [1]
  - Mapping is constructed at this stage and hence immutable
  - Optimization passes preserve precise source-based coverage
- User space usage

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
```

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html#impact-of-llvm-optimizations-on-coverage-reports

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter ⇔ source location**
  - Location includes both line and column
- Instrument at frontend thus **not affected by optimization** [1]
  - Mapping is constructed at this stage and hence immutable
  - Optimization passes preserve precise source-based coverage
- User space usage

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo
```

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html#impact-of-llvm-optimizations-on-coverage-reports

`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter ⇔ source location**
  - Location includes both line and column
- Instrument at frontend thus **not affected by optimization** [1]
  - Mapping is constructed at this stage and hence immutable
  - Optimization passes preserve precise source-based coverage
- User space usage

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo
$ llvm-profdata merge foo.profraw -o foo.profdata
```

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html#impact-of-llvm-optimizations-on-coverage-reports

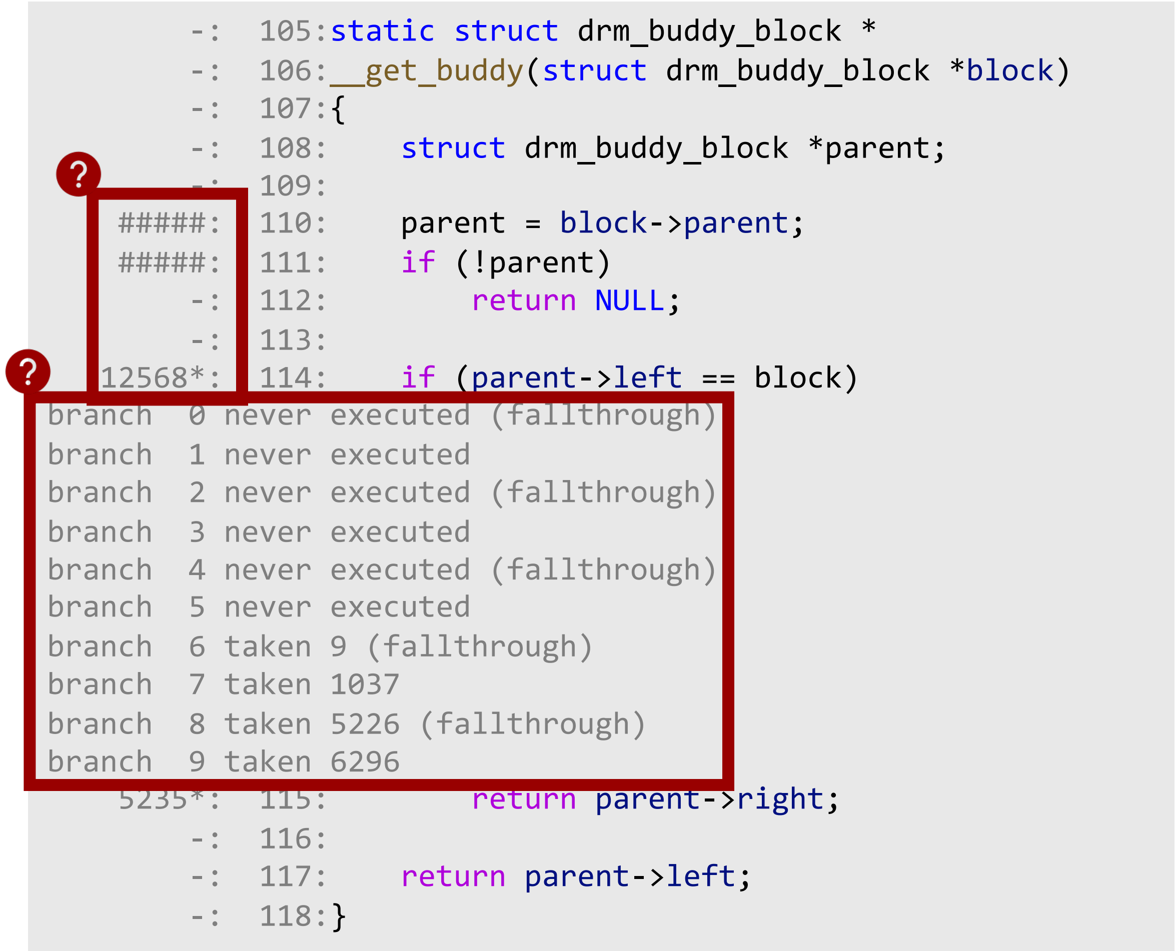
`llvm-cov` and source-based code coverage

- Based on `-fprofile-instr-generate -fcoverage-mapping` flags
- Maintains dedicated mapping **counter ⇔ source location**
  - Location includes both line and column
- Instrument at frontend thus **not affected by optimization** [1]
  - Mapping is constructed at this stage and hence immutable
  - Optimization passes preserve precise source-based coverage
- User space usage

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html#impact-of-llvm-optimizations-on-coverage-reports

Compare gcov and llvm-cov reports

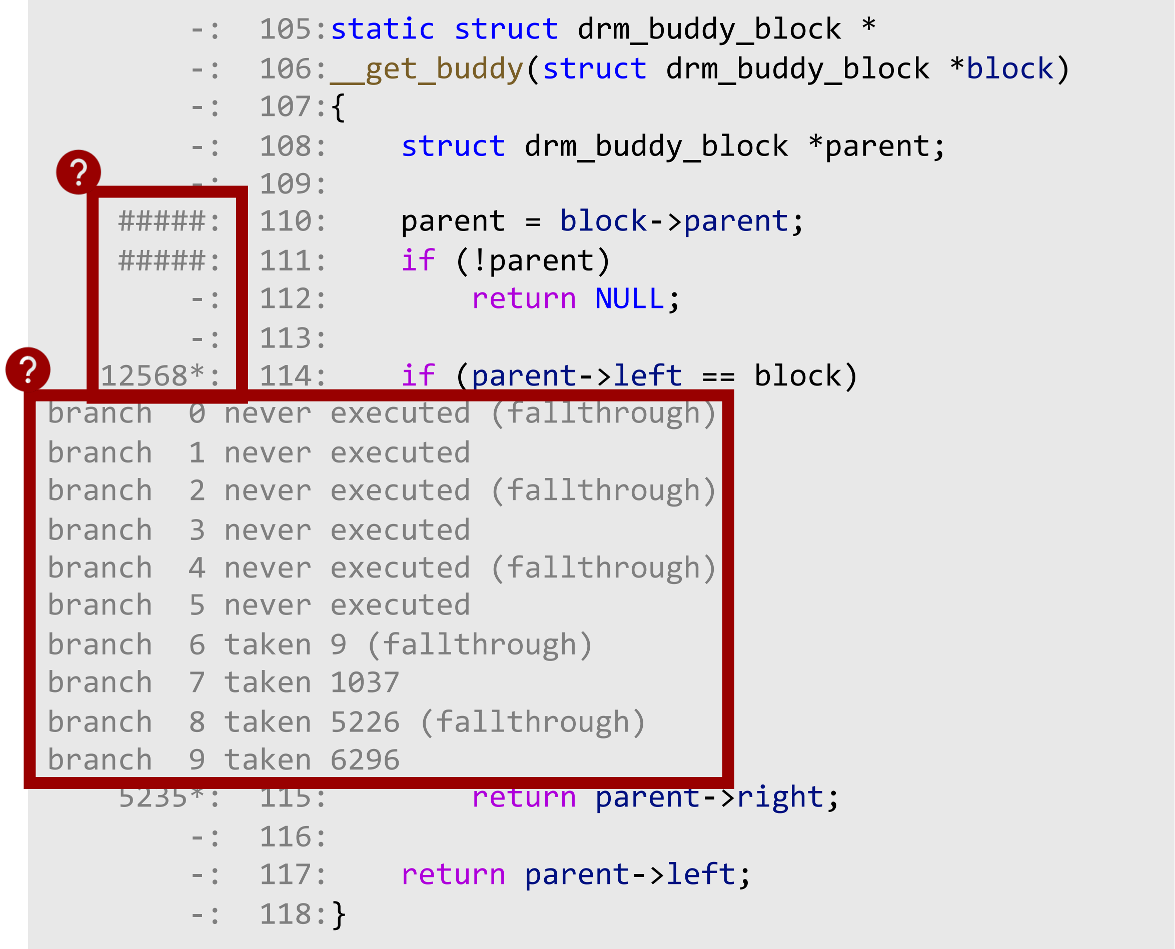


```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

gcov (–O2)

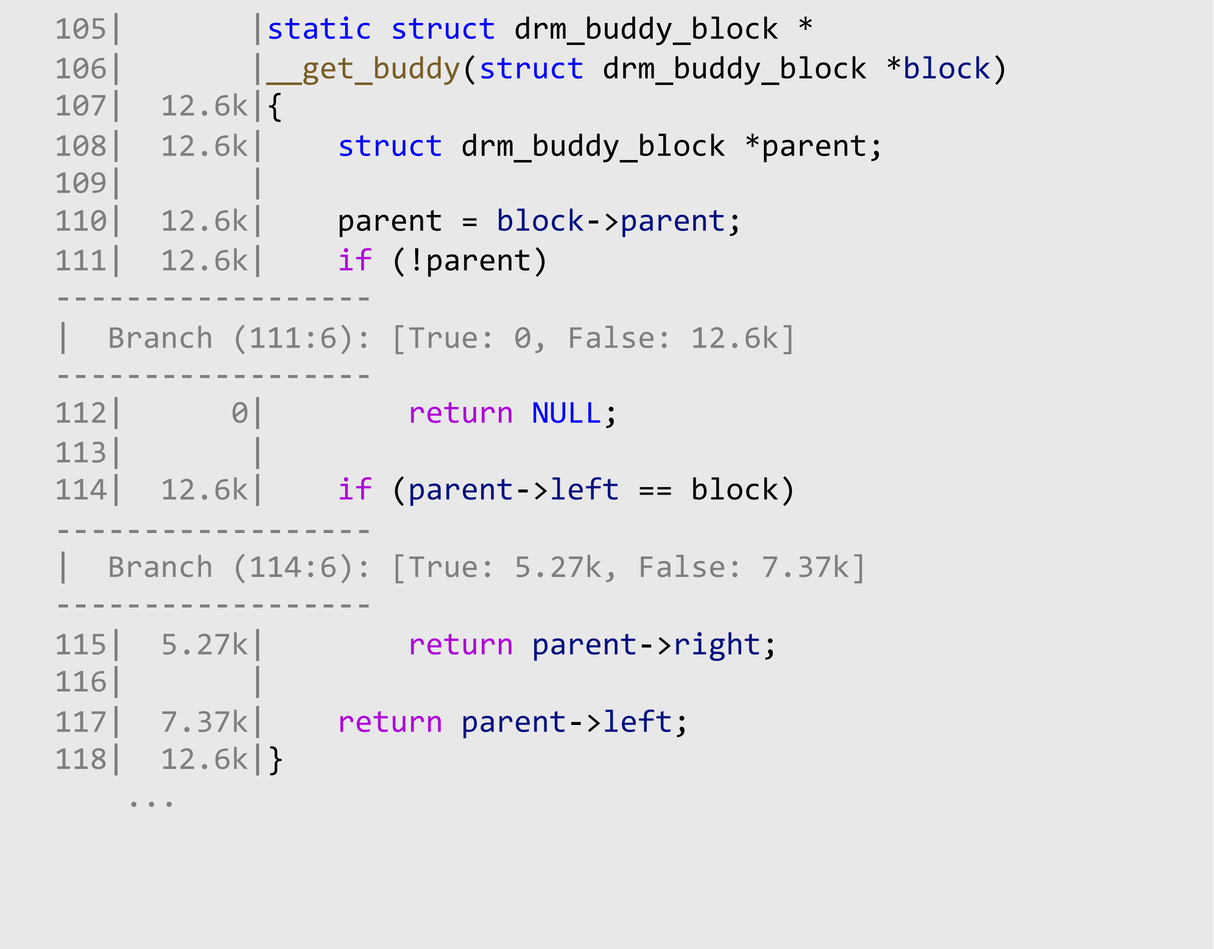
drivers/gpu/drm/drm_buddy.c

Compare gcov and llvm-cov reports



```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

gcov (–O2)

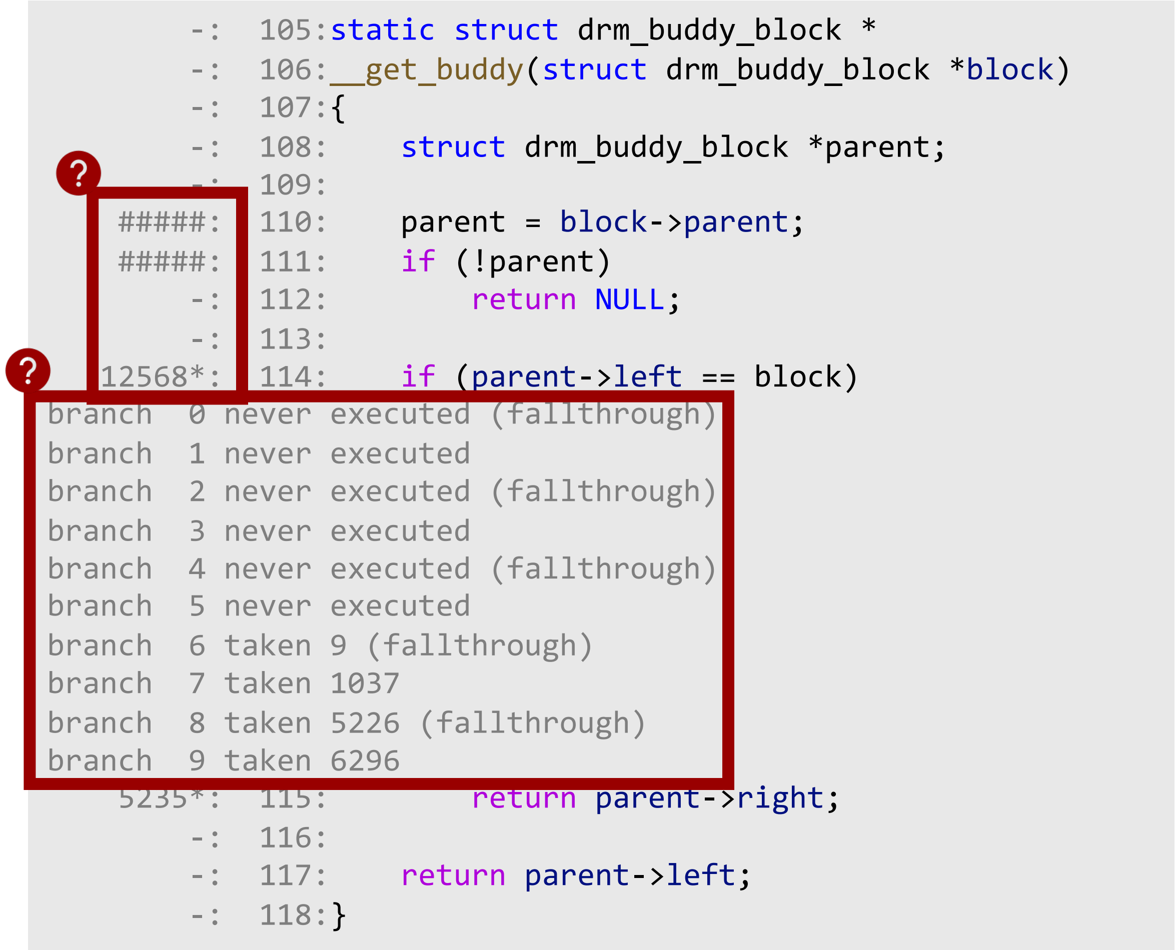


```
  105|       |static struct drm_buddy_block *
  106|       |__get_buddy(struct drm_buddy_block *block)
  107|  12.6k|{
  108|  12.6k|    struct drm_buddy_block *parent;
  109|       |
  110|  12.6k|    parent = block->parent;
  111|  12.6k|    if (!parent)
  ------------------
  |  Branch (111:6): [True: 0, False: 12.6k]
  ------------------
  112|      0|        return NULL;
  113|       |
  114|  12.6k|    if (parent->left == block)
  ------------------
  |  Branch (114:6): [True: 5.27k, False: 7.37k]
  ------------------
  115|  5.27k|        return parent->right;
  116|       |
  117|  7.37k|    return parent->left;
  118|  12.6k|}
     ...
```

llvm-cov (–O2)

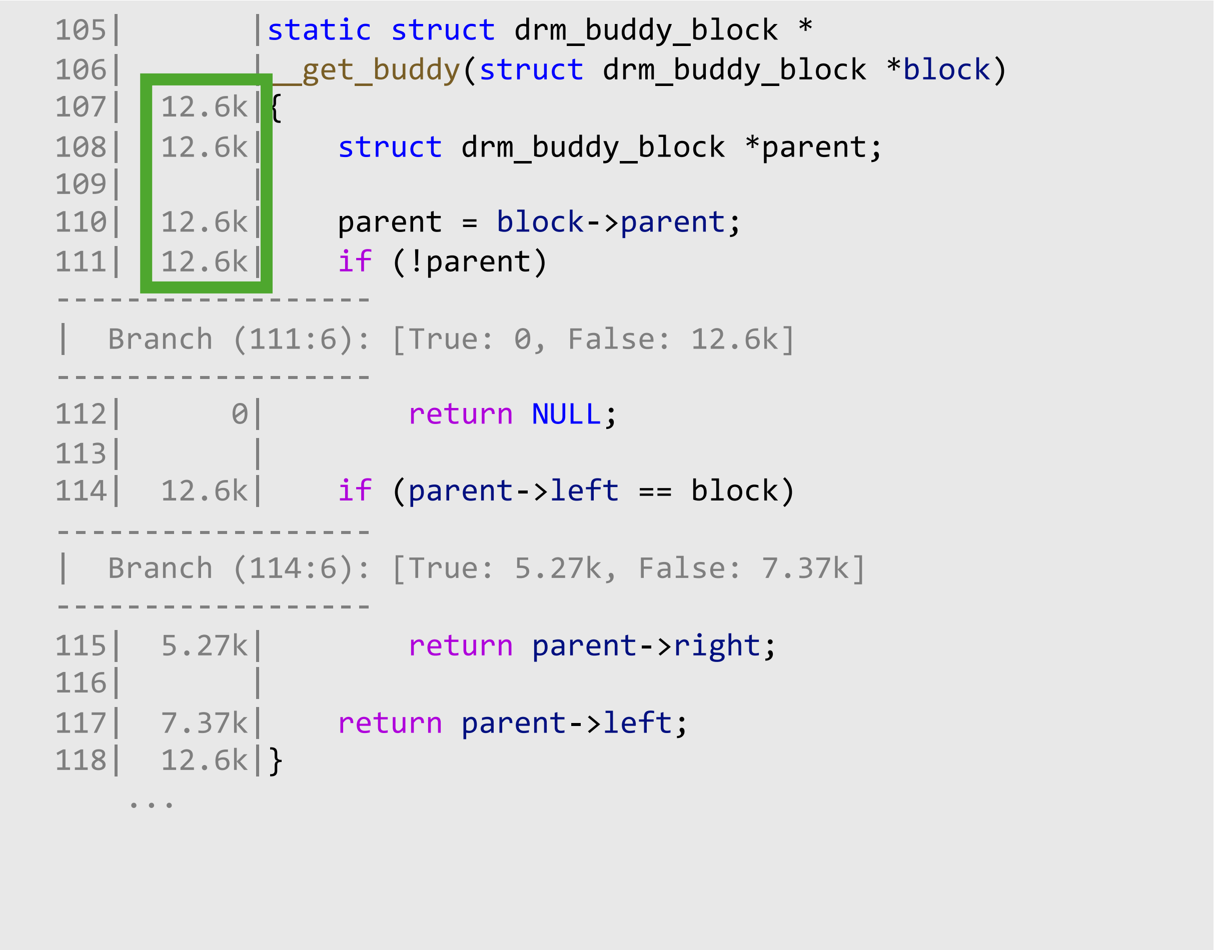
drivers/gpu/drm/drm_buddy.c

Compare gcov and llvm-cov reports



```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

gcov (–O2)

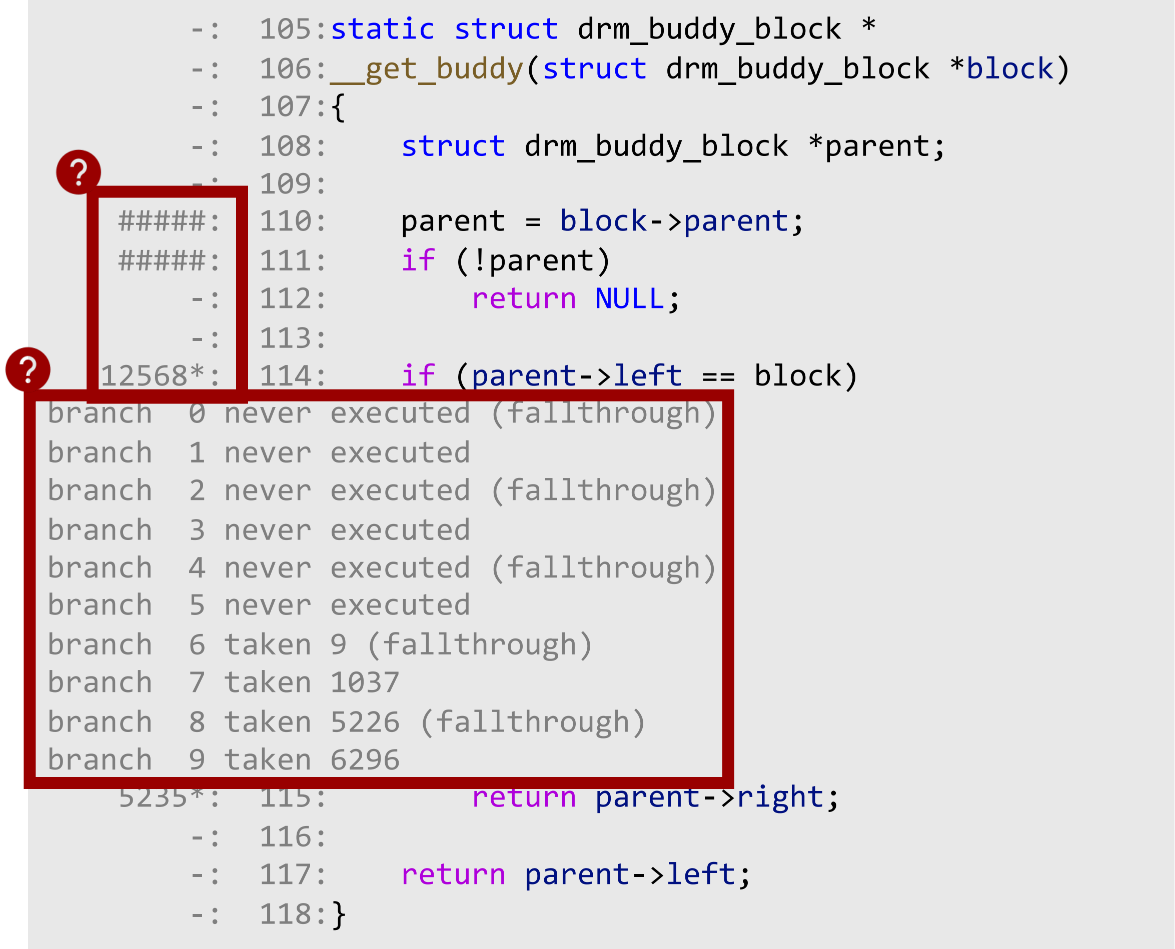


```
  105|       |static struct drm_buddy_block *
  106|       |__get_buddy(struct drm_buddy_block *block)
  107|  12.6k|{
  108|  12.6k|    struct drm_buddy_block *parent;
  109|       |
  110|  12.6k|    parent = block->parent;
  111|  12.6k|    if (!parent)
  ------------------
  |  Branch (111:6): [True: 0, False: 12.6k]
  ------------------
  112|      0|        return NULL;
  113|       |
  114|  12.6k|    if (parent->left == block)
  ------------------
  |  Branch (114:6): [True: 5.27k, False: 7.37k]
  ------------------
  115|  5.27k|        return parent->right;
  116|       |
  117|  7.37k|    return parent->left;
  118|  12.6k|}
      ...
```

llvm-cov (–O2)

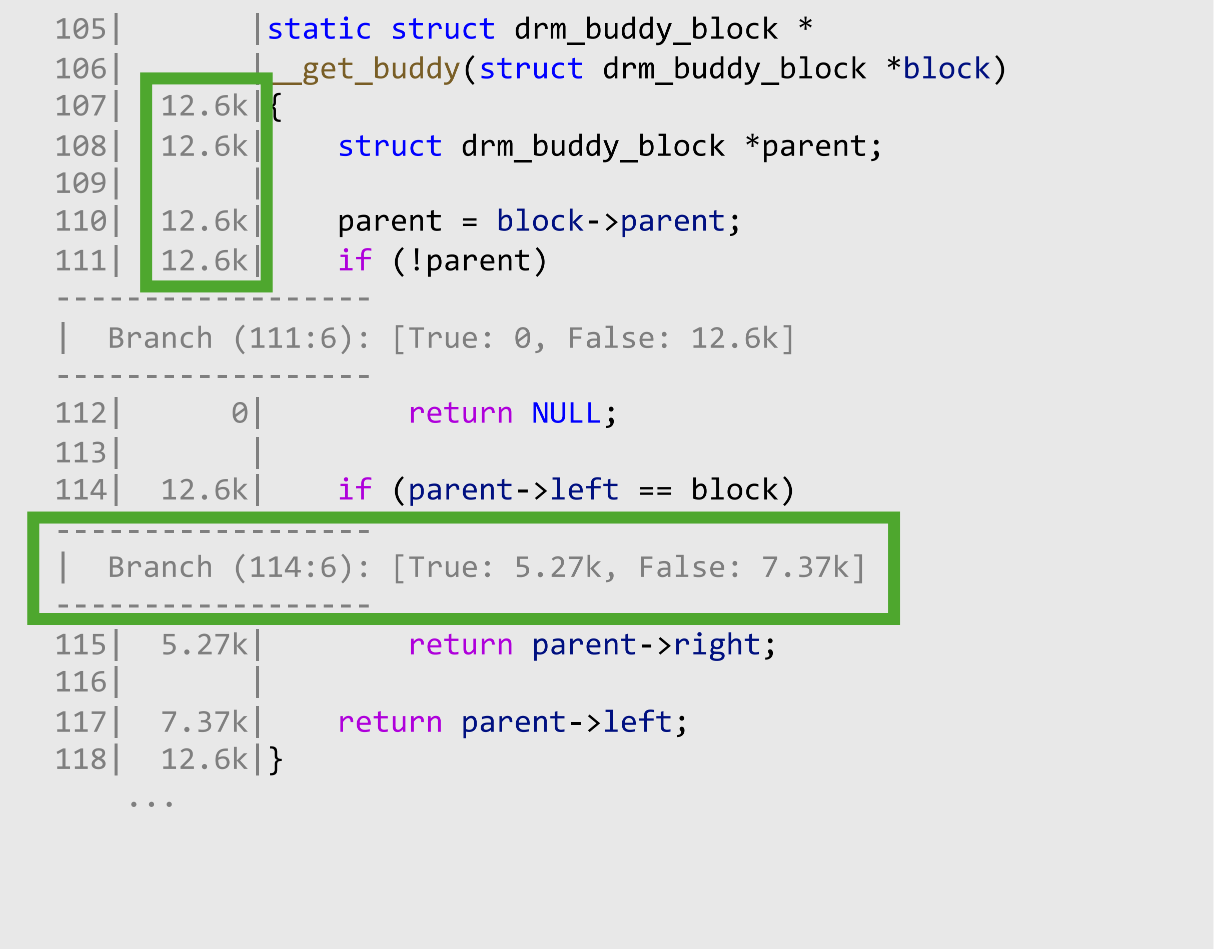
drivers/gpu/drm/drm_buddy.c

Compare gcov and llvm-cov reports



```
        -:  105:static struct drm_buddy_block *
        -:  106:__get_buddy(struct drm_buddy_block *block)
        -:  107:{
        -:  108:    struct drm_buddy_block *parent;
        -:  109:
    #####:  110:    parent = block->parent;
    #####:  111:    if (!parent)
        -:  112:        return NULL;
        -:  113:
   12568*:  114:    if (parent->left == block)
branch  0 never executed (fallthrough)
branch  1 never executed
branch  2 never executed (fallthrough)
branch  3 never executed
branch  4 never executed (fallthrough)
branch  5 never executed
branch  6 taken 9 (fallthrough)
branch  7 taken 1037
branch  8 taken 5226 (fallthrough)
branch  9 taken 6296
    5235*:  115:        return parent->right;
        -:  116:
        -:  117:    return parent->left;
        -:  118:}
```

gcov (–O2)



```
  105|        |static struct drm_buddy_block *
  106|        |__get_buddy(struct drm_buddy_block *block)
  107|   12.6k|{
  108|   12.6k|    struct drm_buddy_block *parent;
  109|        |
  110|   12.6k|    parent = block->parent;
  111|   12.6k|    if (!parent)
  ------------------
  |  Branch (111:6): [True: 0, False: 12.6k]
  ------------------
  112|       0|        return NULL;
  113|        |
  114|   12.6k|    if (parent->left == block)
  ------------------
  |  Branch (114:6): [True: 5.27k, False: 7.37k]
  ------------------
  115|   5.27k|        return parent->right;
  116|        |
  117|   7.37k|    return parent->left;
  118|   12.6k|}
     ...
```

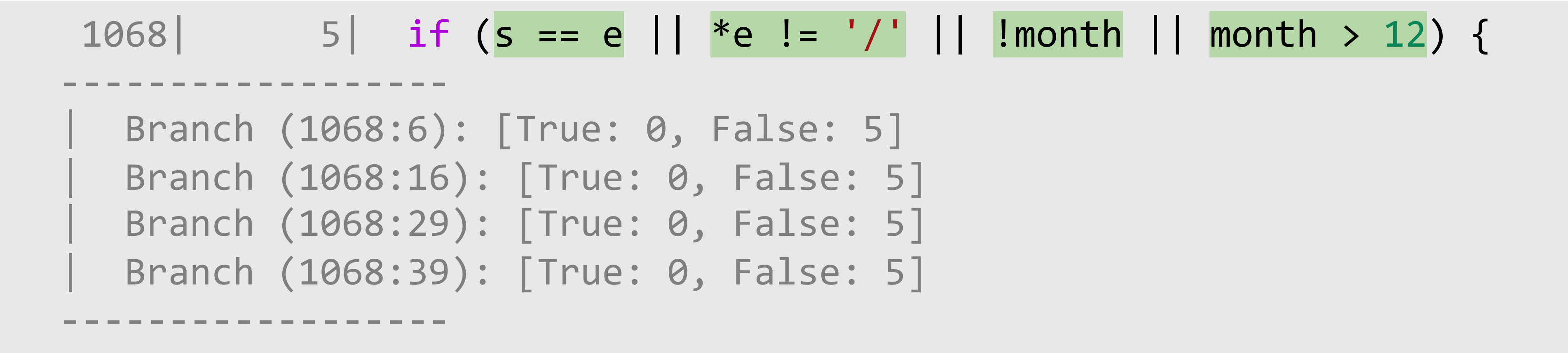
llvm-cov (–O2)

drivers/gpu/drm/drm_buddy.c

More examples: missing branch outcomes

gcov
(-O2)

llvm-cov
(-O2)



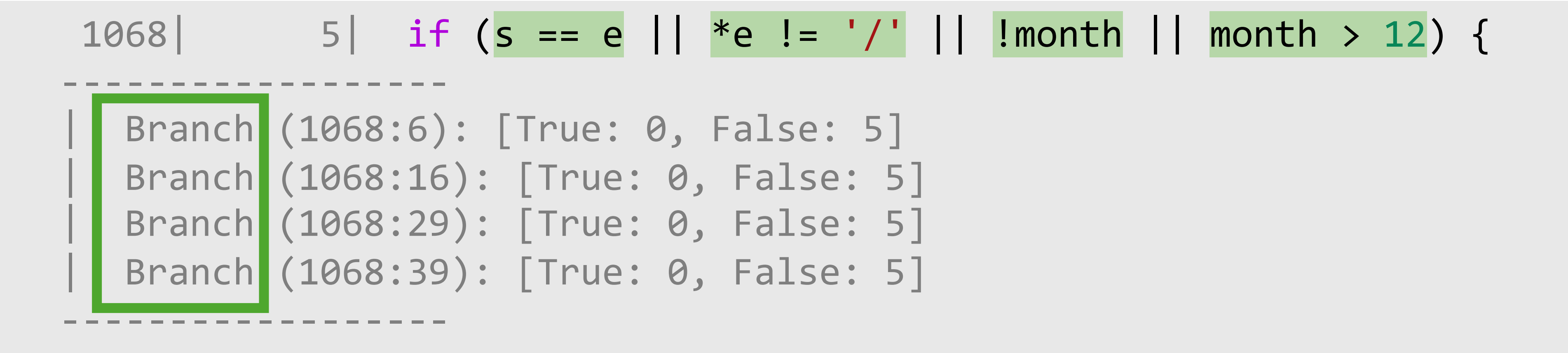
```
 1068|       5|  if (s == e || *e != '/' || !month || month > 12) {
------------------
|  Branch (1068:6): [True: 0, False: 5]
|  Branch (1068:16): [True: 0, False: 5]
|  Branch (1068:29): [True: 0, False: 5]
|  Branch (1068:39): [True: 0, False: 5]
------------------
```

`drivers/firmware/dmi_scan.c:dmi_get_date`

More examples: missing branch outcomes

gcov
(–O2)

llvm-cov
(–O2)



```
 1068|      5|  if (s == e || *e != '/' || !month || month > 12) {
  ------------------
  |  Branch (1068:6): [True: 0, False: 5]
  |  Branch (1068:16): [True: 0, False: 5]
  |  Branch (1068:29): [True: 0, False: 5]
  |  Branch (1068:39): [True: 0, False: 5]
  ------------------
```

`drivers/firmware/dmi_scan.c:dmi_get_date`

More examples: missing branch outcomes

gcov
(–O2)

?

```
        5: 1068:    if (s == e || *e != '/' || !month || month > 12) {
branch  0 taken 5 (fallthrough)
branch  1 taken 0
branch  2 taken 5 (fallthrough)
branch  3 taken 0
branch  4 taken 0 (fallthrough)
branch  5 taken 5
```

llvm-cov
(–O2)

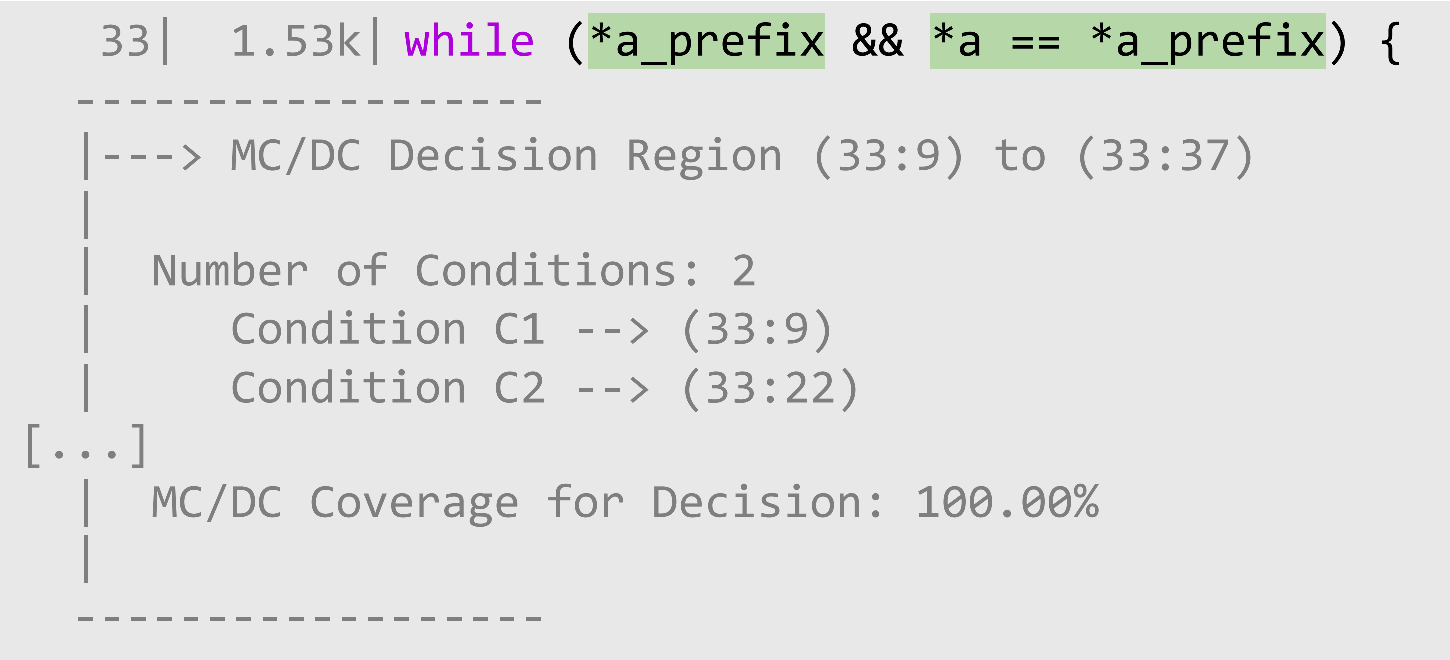
```
 1068|      5|  if (s == e || *e != '/' || !month || month > 12) {
------------------
|  Branch (1068:6): [True: 0, False: 5]
|  Branch (1068:16): [True: 0, False: 5]
|  Branch (1068:29): [True: 0, False: 5]
|  Branch (1068:39): [True: 0, False: 5]
------------------
```

drivers/firmware/dmi_scan.c:dmi_get_date

More examples: MC/DC

gcov
(-O2)

llvm-cov
(-O2)



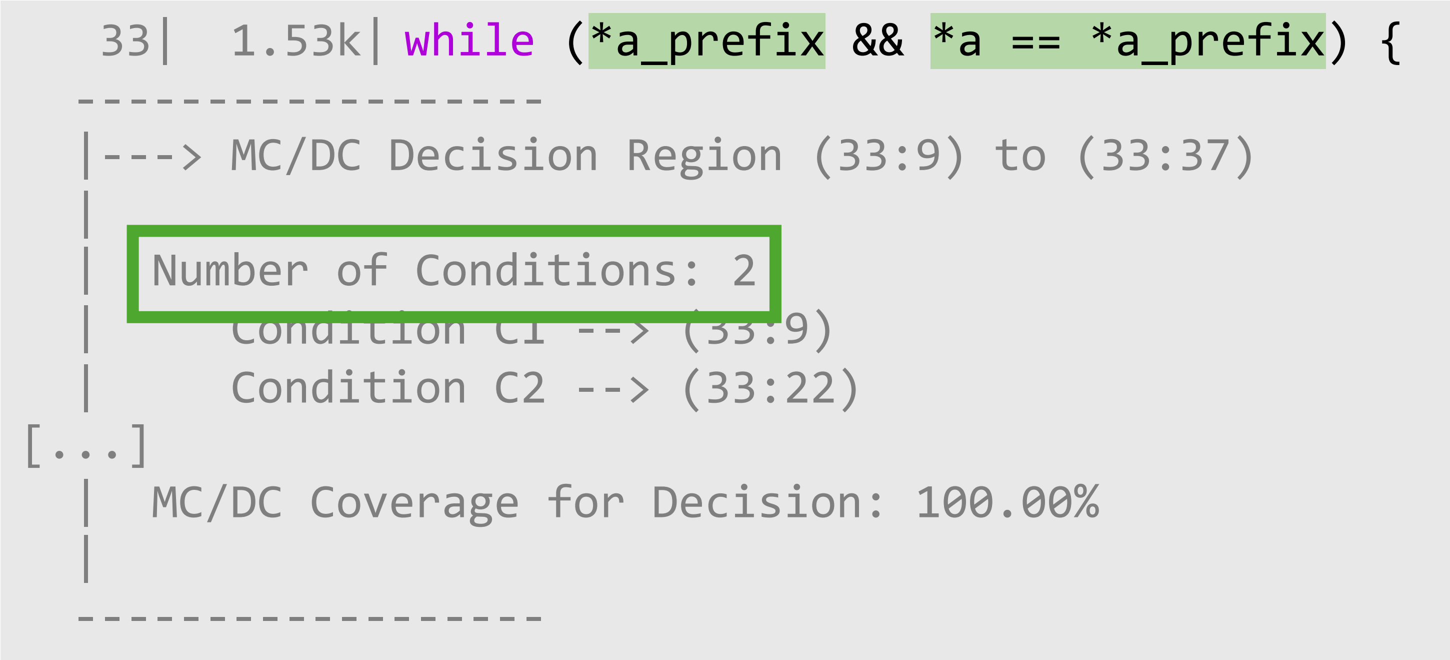
```
    33|  1.53k|while (*a_prefix && *a == *a_prefix) {
  ------------------
  |---> MC/DC Decision Region (33:9) to (33:37)
  |
  |  Number of Conditions: 2
  |     Condition C1 --> (33:9)
  |     Condition C2 --> (33:22)
[...]
  |  MC/DC Coverage for Decision: 100.00%
  |
  ------------------
```

fs/xattr.c:strcmp_prefix

More examples: MC/DC

gcov
(-O2)

llvm-cov
(-O2)

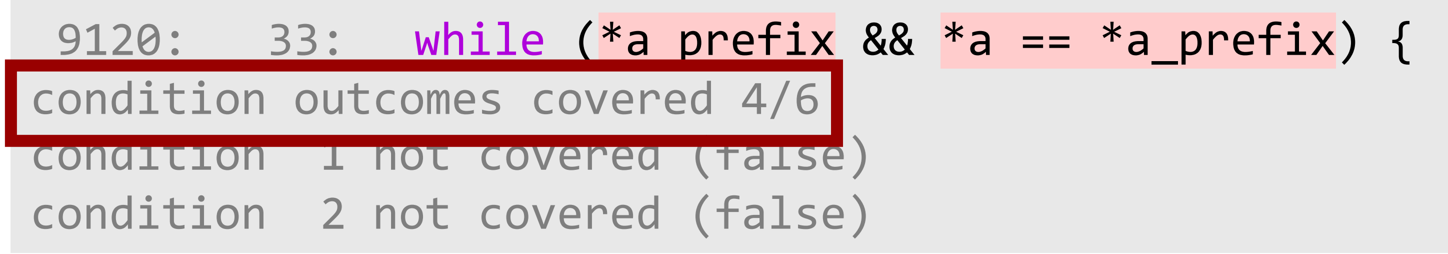


```
    33|  1.53k|while (*a_prefix && *a == *a_prefix) {
  ------------------
  |---> MC/DC Decision Region (33:9) to (33:37)
  |
  |  Number of Conditions: 2
  |     Condition C1 --> (33:9)
  |     Condition C2 --> (33:22)
[...]
  |  MC/DC Coverage for Decision: 100.00%
  |
  ------------------
```

fs/xattr.c:strcmp_prefix

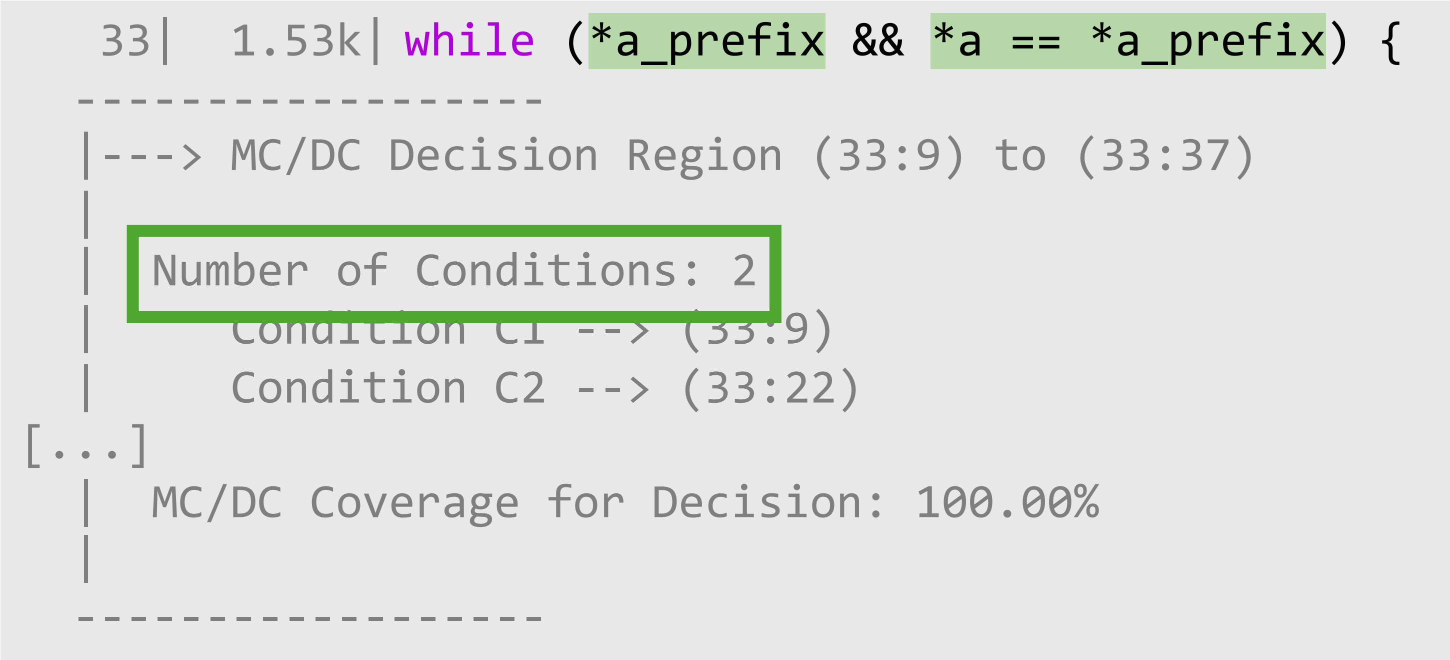
More examples: MC/DC

gcov
(–O2)



```
 9120:    33:   while (*a_prefix && *a == *a_prefix) {
condition outcomes covered 4/6
condition  1 not covered (false)
condition  2 not covered (false)
```

llvm-cov
(–O2)



```
   33|  1.53k| while (*a_prefix && *a == *a_prefix) {
  ------------------
  |---> MC/DC Decision Region (33:9) to (33:37)
  |
  |  Number of Conditions: 2
  |     Condition C1 --> (33:9)
  |     Condition C2 --> (33:22)
[...]
  |  MC/DC Coverage for Decision: 100.00%
  |
  ------------------
```

fs/xattr.c:strcmp_prefix

Proposal of `kernel/llvm-cov/`

Proposal of `kernel/llvm-cov/`

- Source-based code coverage of Linux kernel

Proposal of `kernel/llvm-cov/`

- Source-based code coverage of Linux kernel
- Patches (v2)
  - [1/4] llvm-cov: add Clang's Source-based Code Coverage support
  - [2/4] llvm-cov: add Clang's MC/DC support
  - [3/4] x86: disable llvm-cov instrumentation
  - [4/4] x86: enable llvm-cov support

Proposal of `kernel/llvm-cov/`

- Source-based code coverage of Linux kernel
- Patches (v2)
  - [1/4] llvm-cov: add Clang's Source-based Code Coverage support
  - [2/4] llvm-cov: add Clang's MC/DC support
  - [3/4] x86: disable llvm-cov instrumentation
  - [4/4] x86: enable llvm-cov support
- CI demo https://github.com/xlab-uiuc/linux-mcdc

Patch implementation

Patch implementation

- Kbuild support

Patch implementation

- Kbuild support
  - `CONFIG_LLVM_COV_KERNEL`
  - `CONFIG_LLVM_COV_PROFILE_ALL`

Patch implementation

- Kbuild support
  - `CONFIG_LLVM_COV_KERNEL`
  - `CONFIG_LLVM_COV_PROFILE_ALL`
- Persist raw profiles in a freestanding environment

Patch implementation

- Kbuild support
  - `CONFIG_LLVM_COV_KERNEL`
  - `CONFIG_LLVM_COV_PROFILE_ALL`
- Persist raw profiles in a freestanding environment
  - With no real file system, no runtimes, no libraries, or system calls

Patch implementation

- Kbuild support
  - `CONFIG_LLVM_COV_KERNEL`
  - `CONFIG_LLVM_COV_PROFILE_ALL`
- Persist raw profiles in a freestanding environment
  - With no real file system, no runtimes, no libraries, or system calls
  - Pseudo file system interface
    - `/sys/kernel/debug/llvm-cov/profraw`
    - `/sys/kernel/debug/llvm-cov/reset`

Patch implementation

- Kbuild support
  - `CONFIG_LLVM_COV_KERNEL`
  - `CONFIG_LLVM_COV_PROFILE_ALL`
- Persist raw profiles in a freestanding environment
  - With no real file system, no runtimes, no libraries, or system calls
  - Pseudo file system interface
    - `/sys/kernel/debug/llvm-cov/profraw`
    - `/sys/kernel/debug/llvm-cov/reset`
- Reuse part of patch by Sami Tolvanen et al. "pgo: add clang's Profile Guided Optimization infrastructure patches" [2]

[2] https://lore.kernel.org/lkml/20210407211704.367039-1-morbo@google.com/

Patch implementation

- Kbuild support
  - `CONFIG_LLVM_COV_KERNEL`
  - `CONFIG_LLVM_COV_PROFILE_ALL`
- Persist raw profiles in a freestanding environment
  - With no real file system, no runtimes, no libraries, or system calls
  - Pseudo file system interface
    - `/sys/kernel/debug/llvm-cov/profraw`
    - `/sys/kernel/debug/llvm-cov/reset`
- Reuse part of patch by Sami Tolvanen et al. “pgo: add clang's Profile Guided Optimization infrastructure patches” [2]
  - With different goals: performance optimization vs. **precise coverage** for high assurance

[2] https://lore.kernel.org/lkml/20210407211704.367039-1-morbo@google.com/

Steps of measuring source-based kernel coverage

Steps of measuring source-based kernel coverage

Recap: `llvm-cov` usage in user space

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Steps of measuring source-based kernel coverage

Recap: `llvm-cov` usage in user space

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Steps of measuring source-based kernel coverage

```
$ # Build and instrument the kernel
$ ./scripts/config -e CONFIG_LLVM_COV_KERNEL
$ make LLVM=1
```

Recap: `llvm-cov` usage in user space

⇨

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Steps of measuring source-based kernel coverage

```
$ # Build and instrument the kernel
$ ./scripts/config -e CONFIG_LLVM_COV_KERNEL
$ make LLVM=1
```

Recap: `llvm-cov` usage in user space

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Steps of measuring source-based kernel coverage

```
$ # Build and instrument the kernel
$ ./scripts/config -e CONFIG_LLVM_COV_KERNEL
$ make LLVM=1
```

```
$ # Boot the kernel and run tests
```

Recap: `llvm-cov` usage in user space

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Steps of measuring source-based kernel coverage

```
$ # Build and instrument the kernel
$ ./scripts/config -e CONFIG_LLVM_COV_KERNEL
$ make LLVM=1
```

```
$ # Boot the kernel and run tests
```

```
$ # Generate coverage reports
$ cp /sys/kernel/debug/llvm-cov/profraw vmlinux.profraw
```

Recap: `llvm-cov` usage in user space

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Steps of measuring source-based kernel coverage

```
$ # Build and instrument the kernel
$ ./scripts/config -e CONFIG_LLVM_COV_KERNEL
$ make LLVM=1
```

```
$ # Boot the kernel and run tests
```

```
$ # Generate coverage reports
$ cp /sys/kernel/debug/llvm-cov/profraw vmlinux.profraw
```

Recap: llvm-cov usage in user space

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Steps of measuring source-based kernel coverage

```
$ # Build and instrument the kernel
$ ./scripts/config -e CONFIG_LLVM_COV_KERNEL
$ make LLVM=1
```

```
$ # Boot the kernel and run tests
```

```
$ # Generate coverage reports
$ cp /sys/kernel/debug/llvm-cov/profraw vmlinux.profraw
$ llvm-profdata merge vmlinux.profraw -o vmlinux.profdata
```

Recap: llvm-cov usage in user space

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Steps of measuring source-based kernel coverage

```
$ # Build and instrument the kernel
$ ./scripts/config -e CONFIG_LLVM_COV_KERNEL
$ make LLVM=1
```

```
$ # Boot the kernel and run tests
```

```
$ # Generate coverage reports
$ cp /sys/kernel/debug/llvm-cov/profraw vmlinux.profraw
$ llvm-profdata merge vmlinux.profraw -o vmlinux.profdata
```

Recap: `llvm-cov` usage in user space

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Steps of measuring source-based kernel coverage

```
$ # Build and instrument the kernel
$ ./scripts/config -e CONFIG_LLVM_COV_KERNEL
$ make LLVM=1
```

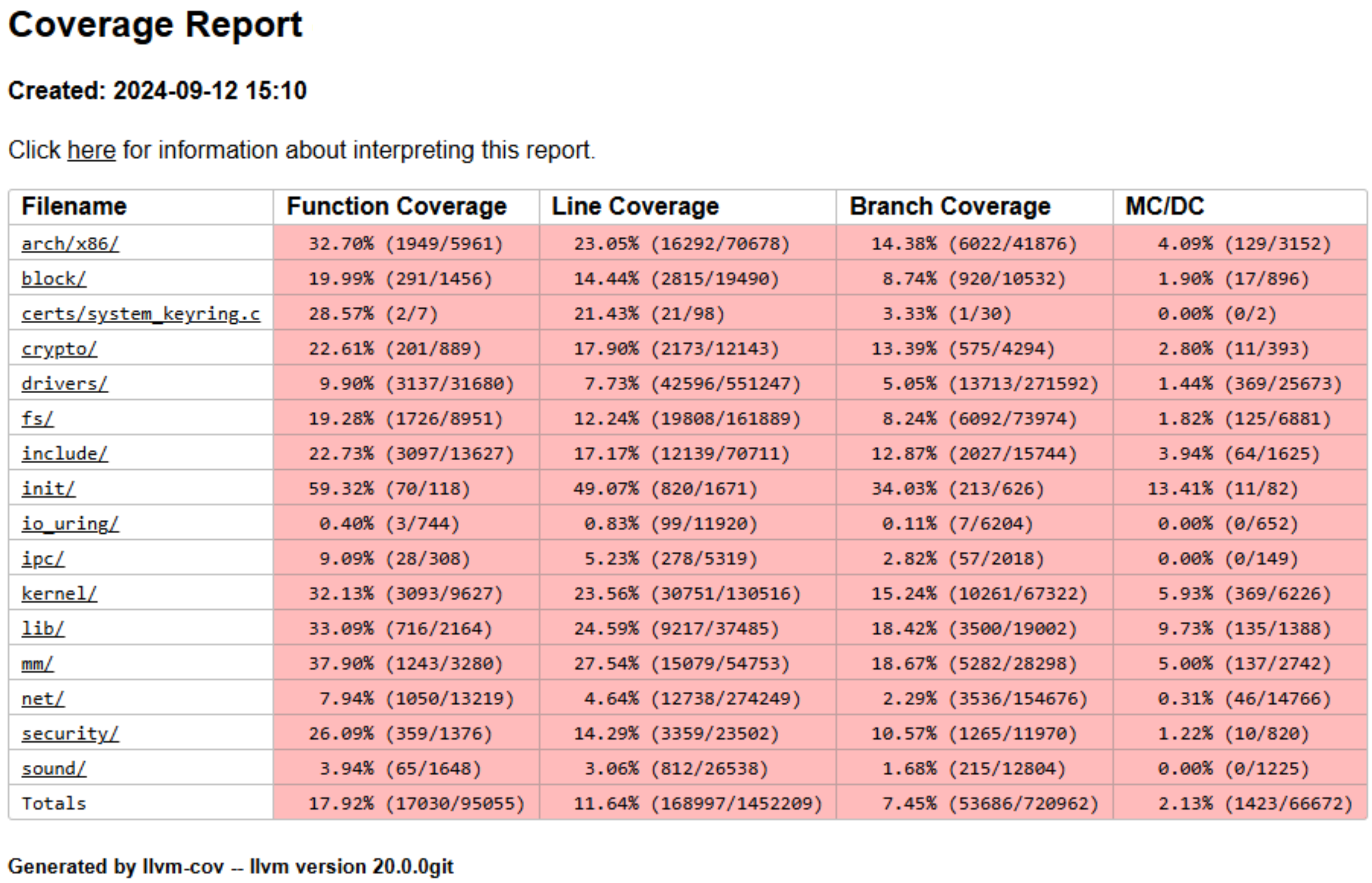
```
$ # Boot the kernel and run tests
```

```
$ # Generate coverage reports
$ cp /sys/kernel/debug/llvm-cov/profraw vmlinux.profraw
$ llvm-profdata merge vmlinux.profraw -o vmlinux.profdata
$ llvm-cov show ./vmlinux -instr-profile=vmlinux.profdata
```

Recap: `llvm-cov` usage in user space

```
$ clang -fprofile-instr-generate -fcoverage-mapping foo.c -o foo
$ LLVM_PROFILE_FILE=foo.profraw ./foo # Run the code and tests
$ llvm-profdata merge foo.profraw -o foo.profdata
$ llvm-cov show ./foo -instr-profile=foo.profdata
```

Example coverage report for KUnit tests



Coverage Report

Created: 2024-09-12 15:10

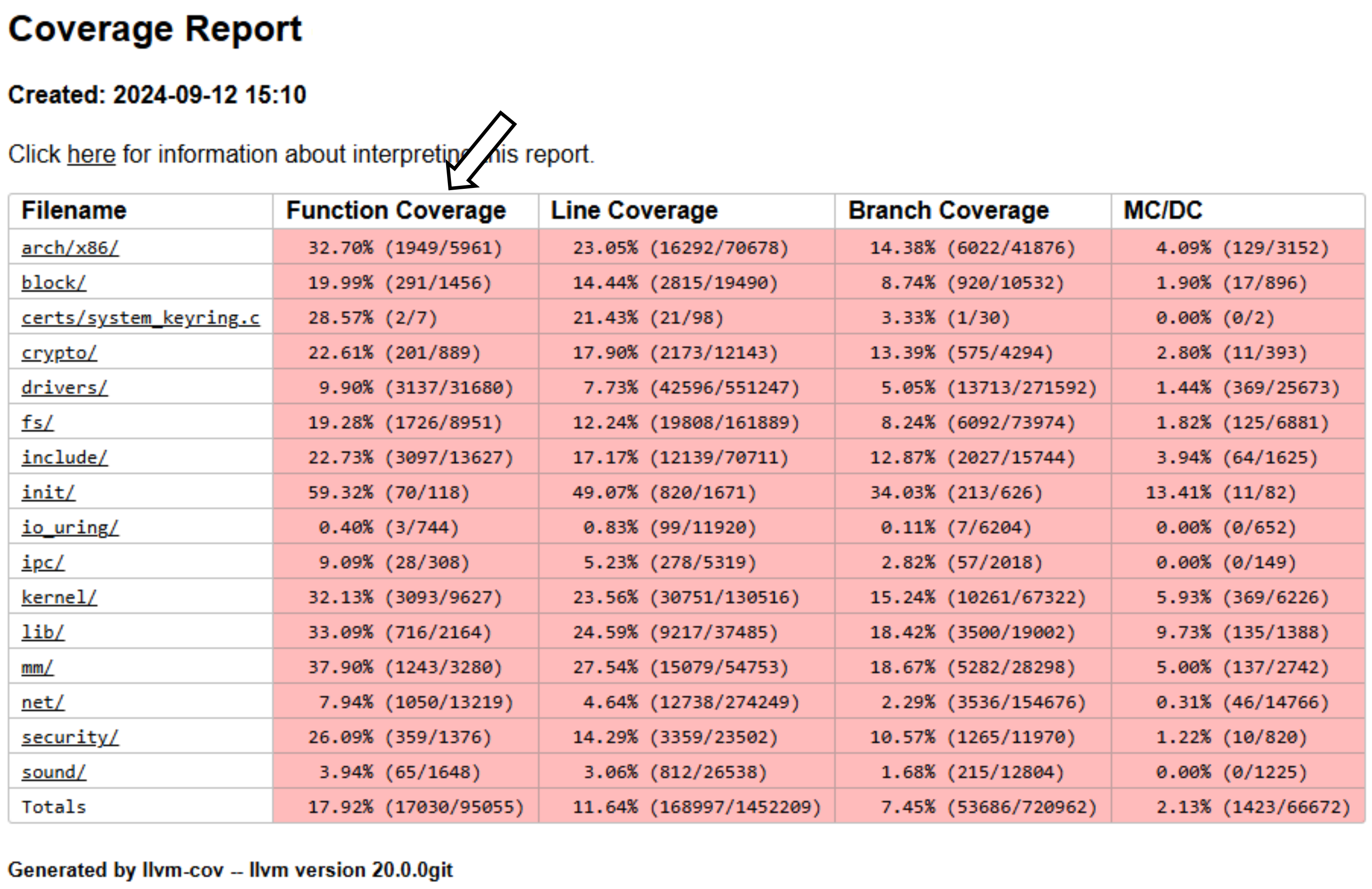
Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Branch Coverage	MC/DC
arch/x86/	32.70% (1949/5961)	23.05% (16292/70678)	14.38% (6022/41876)	4.09% (129/3152)
block/	19.99% (291/1456)	14.44% (2815/19490)	8.74% (920/10532)	1.90% (17/896)
certs/system_keyring.c	28.57% (2/7)	21.43% (21/98)	3.33% (1/30)	0.00% (0/2)
crypto/	22.61% (201/889)	17.90% (2173/12143)	13.39% (575/4294)	2.80% (11/393)
drivers/	9.90% (3137/31680)	7.73% (42596/551247)	5.05% (13713/271592)	1.44% (369/25673)
fs/	19.28% (1726/8951)	12.24% (19808/161889)	8.24% (6092/73974)	1.82% (125/6881)
include/	22.73% (3097/13627)	17.17% (12139/70711)	12.87% (2027/15744)	3.94% (64/1625)
init/	59.32% (70/118)	49.07% (820/1671)	34.03% (213/626)	13.41% (11/82)
io_uring/	0.40% (3/744)	0.83% (99/11920)	0.11% (7/6204)	0.00% (0/652)
ipc/	9.09% (28/308)	5.23% (278/5319)	2.82% (57/2018)	0.00% (0/149)
kernel/	32.13% (3093/9627)	23.56% (30751/130516)	15.24% (10261/67322)	5.93% (369/6226)
lib/	33.09% (716/2164)	24.59% (9217/37485)	18.42% (3500/19002)	9.73% (135/1388)
mm/	37.90% (1243/3280)	27.54% (15079/54753)	18.67% (5282/28298)	5.00% (137/2742)
net/	7.94% (1050/13219)	4.64% (12738/274249)	2.29% (3536/154676)	0.31% (46/14766)
security/	26.09% (359/1376)	14.29% (3359/23502)	10.57% (1265/11970)	1.22% (10/820)
sound/	3.94% (65/1648)	3.06% (812/26538)	1.68% (215/12804)	0.00% (0/1225)
Totals	17.92% (17030/95055)	11.64% (168997/1452209)	7.45% (53686/720962)	2.13% (1423/66672)

Generated by llvm-cov -- llvm version 20.0.0git

Summary page

Example coverage report for KUnit tests



Coverage Report

Created: 2024-09-12 15:10

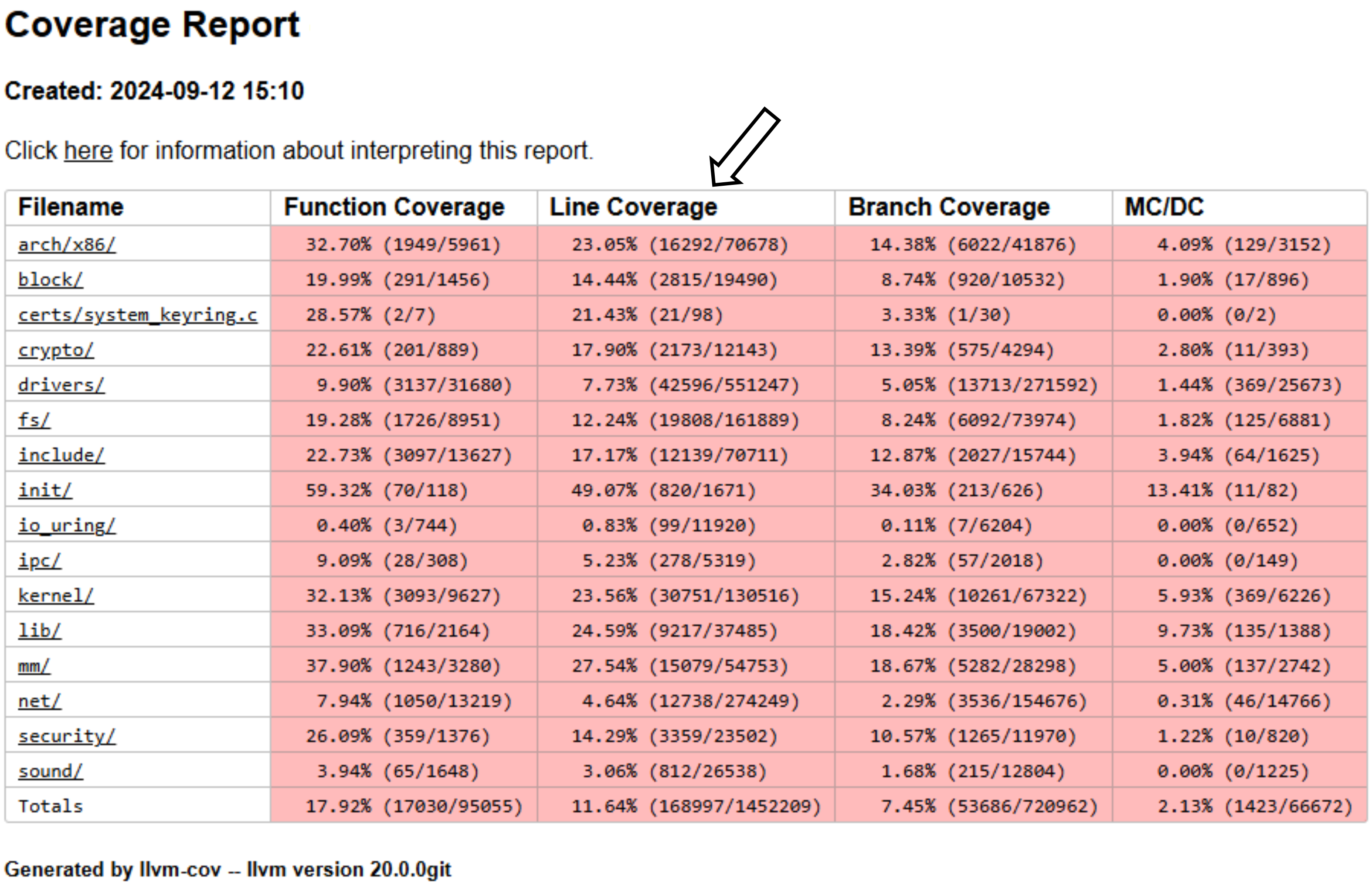
Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Branch Coverage	MC/DC
arch/x86/	32.70% (1949/5961)	23.05% (16292/70678)	14.38% (6022/41876)	4.09% (129/3152)
block/	19.99% (291/1456)	14.44% (2815/19490)	8.74% (920/10532)	1.90% (17/896)
certs/system_keyring.c	28.57% (2/7)	21.43% (21/98)	3.33% (1/30)	0.00% (0/2)
crypto/	22.61% (201/889)	17.90% (2173/12143)	13.39% (575/4294)	2.80% (11/393)
drivers/	9.90% (3137/31680)	7.73% (42596/551247)	5.05% (13713/271592)	1.44% (369/25673)
fs/	19.28% (1726/8951)	12.24% (19808/161889)	8.24% (6092/73974)	1.82% (125/6881)
include/	22.73% (3097/13627)	17.17% (12139/70711)	12.87% (2027/15744)	3.94% (64/1625)
init/	59.32% (70/118)	49.07% (820/1671)	34.03% (213/626)	13.41% (11/82)
io_uring/	0.40% (3/744)	0.83% (99/11920)	0.11% (7/6204)	0.00% (0/652)
ipc/	9.09% (28/308)	5.23% (278/5319)	2.82% (57/2018)	0.00% (0/149)
kernel/	32.13% (3093/9627)	23.56% (30751/130516)	15.24% (10261/67322)	5.93% (369/6226)
lib/	33.09% (716/2164)	24.59% (9217/37485)	18.42% (3500/19002)	9.73% (135/1388)
mm/	37.90% (1243/3280)	27.54% (15079/54753)	18.67% (5282/28298)	5.00% (137/2742)
net/	7.94% (1050/13219)	4.64% (12738/274249)	2.29% (3536/154676)	0.31% (46/14766)
security/	26.09% (359/1376)	14.29% (3359/23502)	10.57% (1265/11970)	1.22% (10/820)
sound/	3.94% (65/1648)	3.06% (812/26538)	1.68% (215/12804)	0.00% (0/1225)
Totals	17.92% (17030/95055)	11.64% (168997/1452209)	7.45% (53686/720962)	2.13% (1423/66672)

Generated by llvm-cov -- llvm version 20.0.0git

Summary page

Example coverage report for KUnit tests



Coverage Report

Created: 2024-09-12 15:10

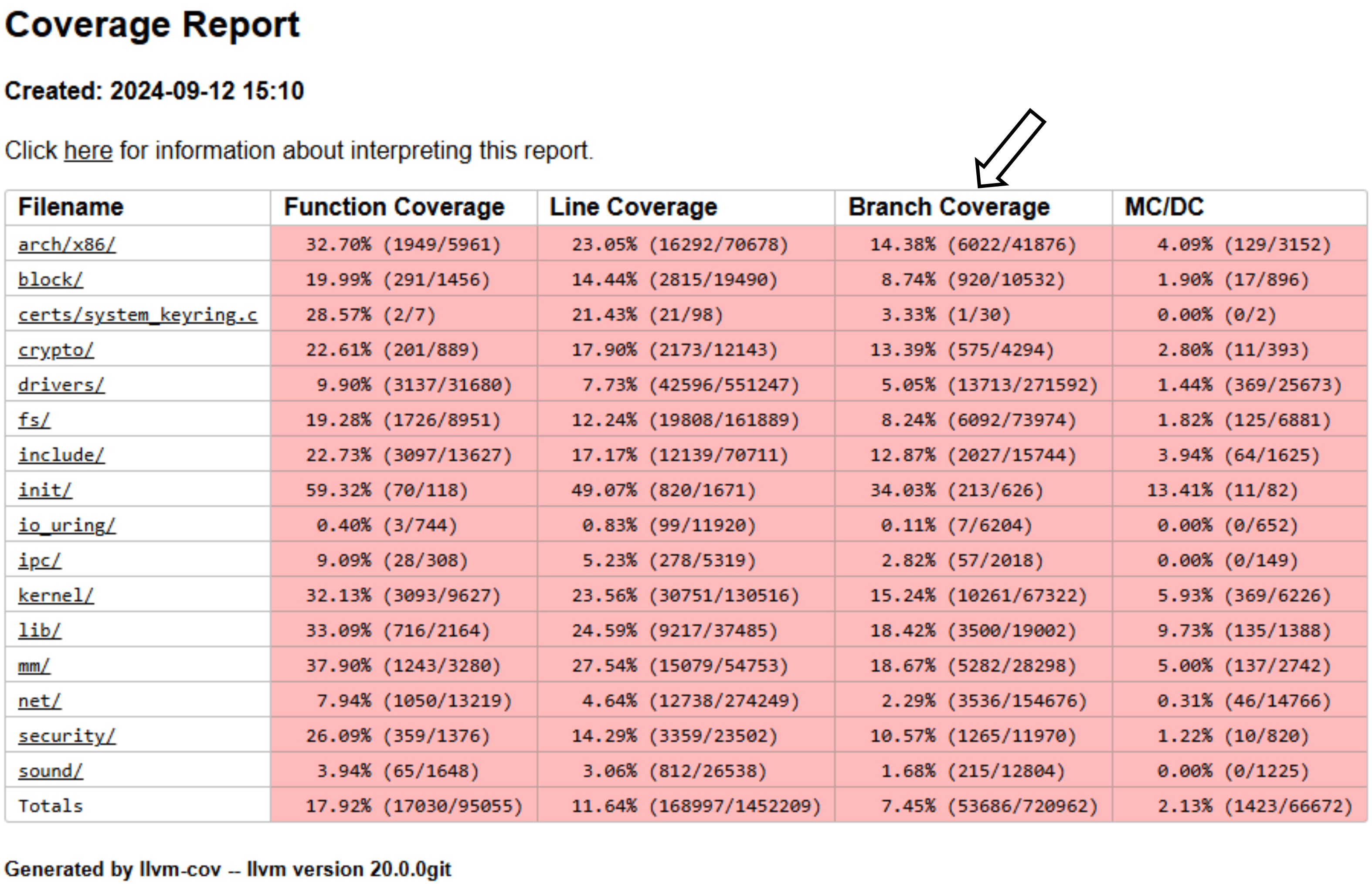
Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Branch Coverage	MC/DC
arch/x86/	32.70% (1949/5961)	23.05% (16292/70678)	14.38% (6022/41876)	4.09% (129/3152)
block/	19.99% (291/1456)	14.44% (2815/19490)	8.74% (920/10532)	1.90% (17/896)
certs/system_keyring.c	28.57% (2/7)	21.43% (21/98)	3.33% (1/30)	0.00% (0/2)
crypto/	22.61% (201/889)	17.90% (2173/12143)	13.39% (575/4294)	2.80% (11/393)
drivers/	9.90% (3137/31680)	7.73% (42596/551247)	5.05% (13713/271592)	1.44% (369/25673)
fs/	19.28% (1726/8951)	12.24% (19808/161889)	8.24% (6092/73974)	1.82% (125/6881)
include/	22.73% (3097/13627)	17.17% (12139/70711)	12.87% (2027/15744)	3.94% (64/1625)
init/	59.32% (70/118)	49.07% (820/1671)	34.03% (213/626)	13.41% (11/82)
io_uring/	0.40% (3/744)	0.83% (99/11920)	0.11% (7/6204)	0.00% (0/652)
ipc/	9.09% (28/308)	5.23% (278/5319)	2.82% (57/2018)	0.00% (0/149)
kernel/	32.13% (3093/9627)	23.56% (30751/130516)	15.24% (10261/67322)	5.93% (369/6226)
lib/	33.09% (716/2164)	24.59% (9217/37485)	18.42% (3500/19002)	9.73% (135/1388)
mm/	37.90% (1243/3280)	27.54% (15079/54753)	18.67% (5282/28298)	5.00% (137/2742)
net/	7.94% (1050/13219)	4.64% (12738/274249)	2.29% (3536/154676)	0.31% (46/14766)
security/	26.09% (359/1376)	14.29% (3359/23502)	10.57% (1265/11970)	1.22% (10/820)
sound/	3.94% (65/1648)	3.06% (812/26538)	1.68% (215/12804)	0.00% (0/1225)
Totals	17.92% (17030/95055)	11.64% (168997/1452209)	7.45% (53686/720962)	2.13% (1423/66672)

Generated by llvm-cov -- llvm version 20.0.0git

Summary page

Example coverage report for KUnit tests



Coverage Report

Created: 2024-09-12 15:10

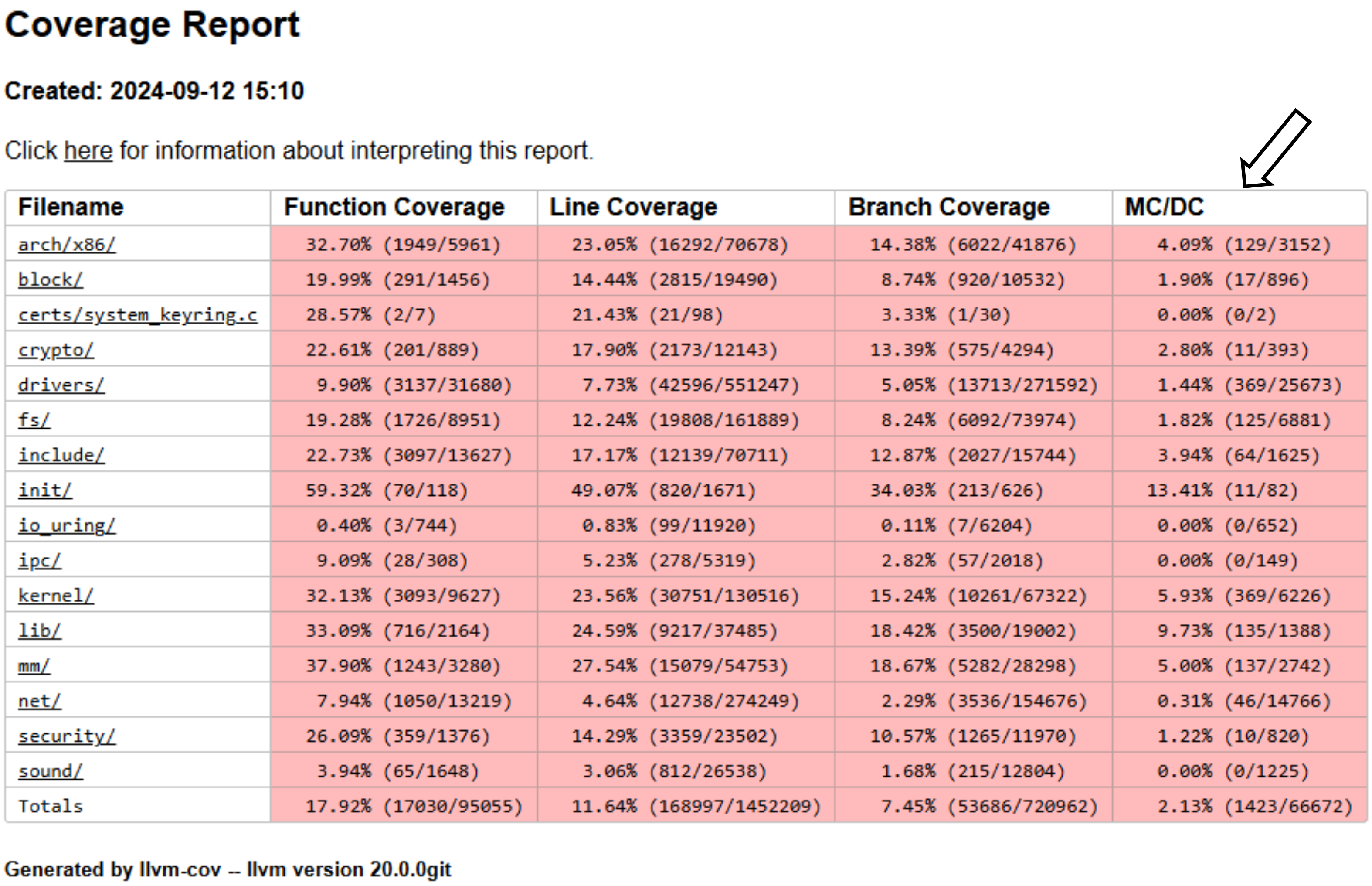
Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Branch Coverage	MC/DC
arch/x86/	32.70% (1949/5961)	23.05% (16292/70678)	14.38% (6022/41876)	4.09% (129/3152)
block/	19.99% (291/1456)	14.44% (2815/19490)	8.74% (920/10532)	1.90% (17/896)
certs/system_keyring.c	28.57% (2/7)	21.43% (21/98)	3.33% (1/30)	0.00% (0/2)
crypto/	22.61% (201/889)	17.90% (2173/12143)	13.39% (575/4294)	2.80% (11/393)
drivers/	9.90% (3137/31680)	7.73% (42596/551247)	5.05% (13713/271592)	1.44% (369/25673)
fs/	19.28% (1726/8951)	12.24% (19808/161889)	8.24% (6092/73974)	1.82% (125/6881)
include/	22.73% (3097/13627)	17.17% (12139/70711)	12.87% (2027/15744)	3.94% (64/1625)
init/	59.32% (70/118)	49.07% (820/1671)	34.03% (213/626)	13.41% (11/82)
io_uring/	0.40% (3/744)	0.83% (99/11920)	0.11% (7/6204)	0.00% (0/652)
ipc/	9.09% (28/308)	5.23% (278/5319)	2.82% (57/2018)	0.00% (0/149)
kernel/	32.13% (3093/9627)	23.56% (30751/130516)	15.24% (10261/67322)	5.93% (369/6226)
lib/	33.09% (716/2164)	24.59% (9217/37485)	18.42% (3500/19002)	9.73% (135/1388)
mm/	37.90% (1243/3280)	27.54% (15079/54753)	18.67% (5282/28298)	5.00% (137/2742)
net/	7.94% (1050/13219)	4.64% (12738/274249)	2.29% (3536/154676)	0.31% (46/14766)
security/	26.09% (359/1376)	14.29% (3359/23502)	10.57% (1265/11970)	1.22% (10/820)
sound/	3.94% (65/1648)	3.06% (812/26538)	1.68% (215/12804)	0.00% (0/1225)
Totals	17.92% (17030/95055)	11.64% (168997/1452209)	7.45% (53686/720962)	2.13% (1423/66672)

Generated by llvm-cov -- llvm version 20.0.0git

Summary page

Example coverage report for KUnit tests



Coverage Report

Created: 2024-09-12 15:10

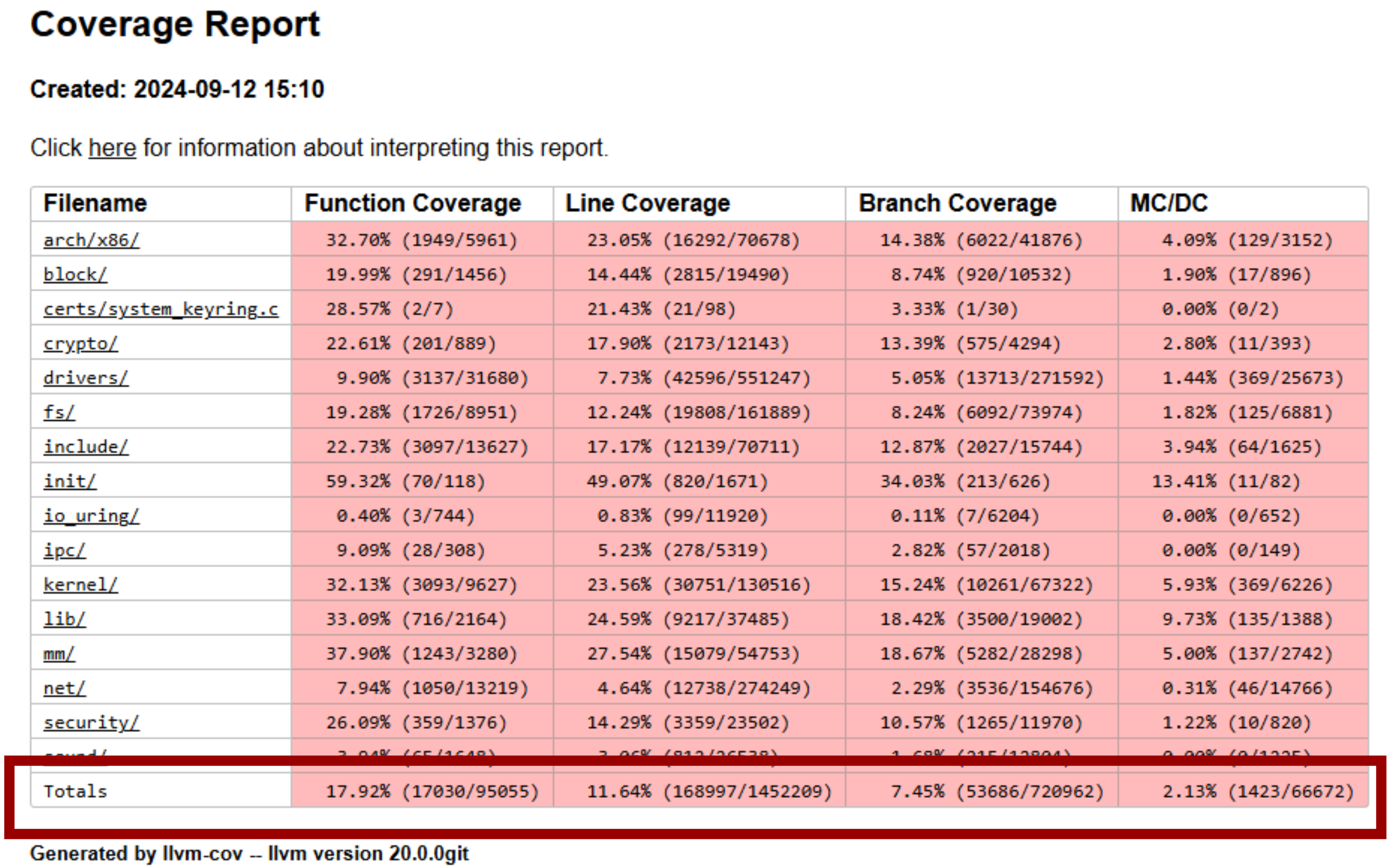
Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Branch Coverage	MC/DC
arch/x86/	32.70% (1949/5961)	23.05% (16292/70678)	14.38% (6022/41876)	4.09% (129/3152)
block/	19.99% (291/1456)	14.44% (2815/19490)	8.74% (920/10532)	1.90% (17/896)
certs/system_keyring.c	28.57% (2/7)	21.43% (21/98)	3.33% (1/30)	0.00% (0/2)
crypto/	22.61% (201/889)	17.90% (2173/12143)	13.39% (575/4294)	2.80% (11/393)
drivers/	9.90% (3137/31680)	7.73% (42596/551247)	5.05% (13713/271592)	1.44% (369/25673)
fs/	19.28% (1726/8951)	12.24% (19808/161889)	8.24% (6092/73974)	1.82% (125/6881)
include/	22.73% (3097/13627)	17.17% (12139/70711)	12.87% (2027/15744)	3.94% (64/1625)
init/	59.32% (70/118)	49.07% (820/1671)	34.03% (213/626)	13.41% (11/82)
io_uring/	0.40% (3/744)	0.83% (99/11920)	0.11% (7/6204)	0.00% (0/652)
ipc/	9.09% (28/308)	5.23% (278/5319)	2.82% (57/2018)	0.00% (0/149)
kernel/	32.13% (3093/9627)	23.56% (30751/130516)	15.24% (10261/67322)	5.93% (369/6226)
lib/	33.09% (716/2164)	24.59% (9217/37485)	18.42% (3500/19002)	9.73% (135/1388)
mm/	37.90% (1243/3280)	27.54% (15079/54753)	18.67% (5282/28298)	5.00% (137/2742)
net/	7.94% (1050/13219)	4.64% (12738/274249)	2.29% (3536/154676)	0.31% (46/14766)
security/	26.09% (359/1376)	14.29% (3359/23502)	10.57% (1265/11970)	1.22% (10/820)
sound/	3.94% (65/1648)	3.06% (812/26538)	1.68% (215/12804)	0.00% (0/1225)
Totals	17.92% (17030/95055)	11.64% (168997/1452209)	7.45% (53686/720962)	2.13% (1423/66672)

Generated by llvm-cov -- llvm version 20.0.0git

Summary page

Example coverage report for KUnit tests



Coverage Report

Created: 2024-09-12 15:10

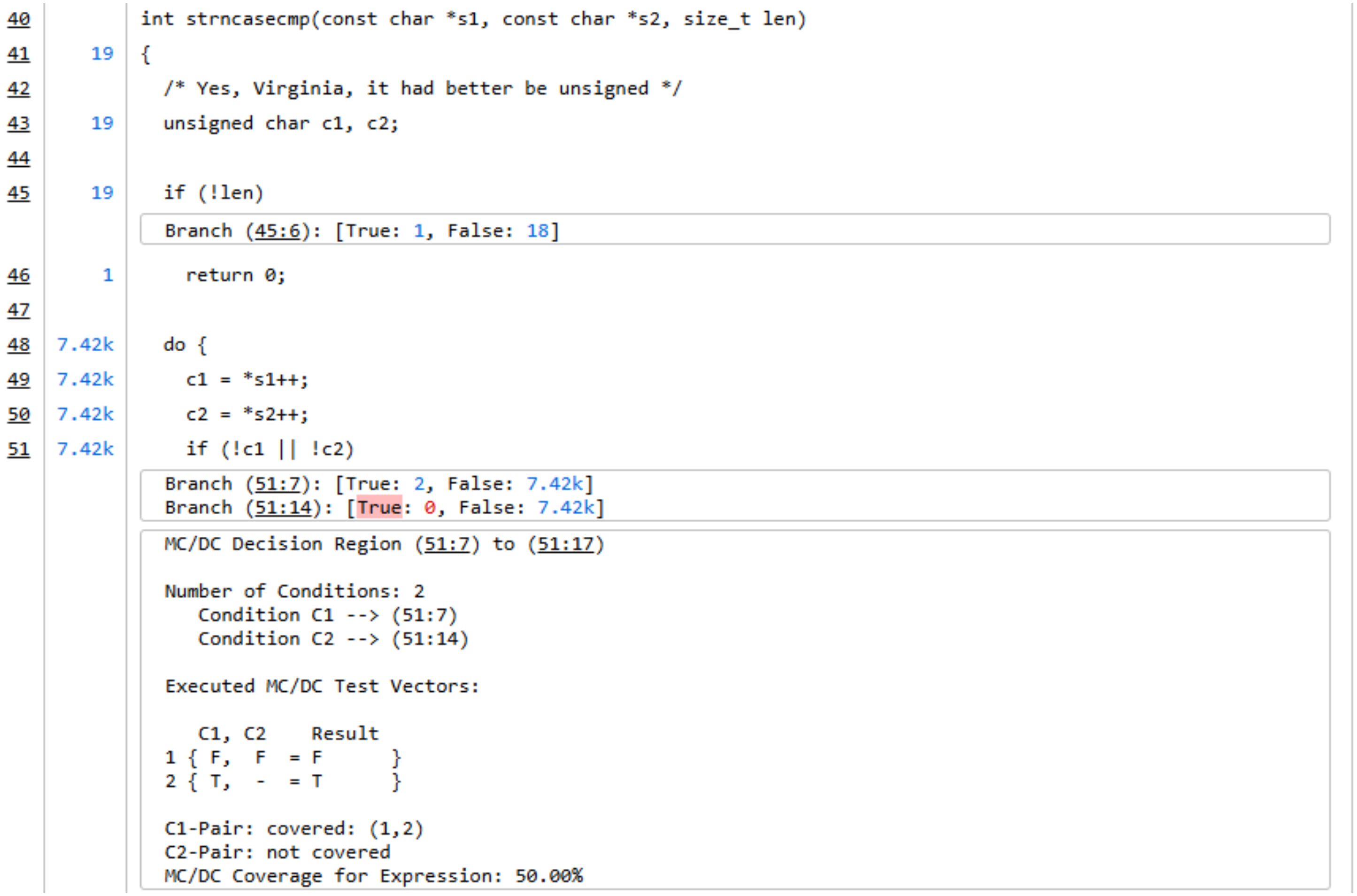
Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Branch Coverage	MC/DC
arch/x86/	32.70% (1949/5961)	23.05% (16292/70678)	14.38% (6022/41876)	4.09% (129/3152)
block/	19.99% (291/1456)	14.44% (2815/19490)	8.74% (920/10532)	1.90% (17/896)
certs/system_keyring.c	28.57% (2/7)	21.43% (21/98)	3.33% (1/30)	0.00% (0/2)
crypto/	22.61% (201/889)	17.90% (2173/12143)	13.39% (575/4294)	2.80% (11/393)
drivers/	9.90% (3137/31680)	7.73% (42596/551247)	5.05% (13713/271592)	1.44% (369/25673)
fs/	19.28% (1726/8951)	12.24% (19808/161889)	8.24% (6092/73974)	1.82% (125/6881)
include/	22.73% (3097/13627)	17.17% (12139/70711)	12.87% (2027/15744)	3.94% (64/1625)
init/	59.32% (70/118)	49.07% (820/1671)	34.03% (213/626)	13.41% (11/82)
io_uring/	0.40% (3/744)	0.83% (99/11920)	0.11% (7/6204)	0.00% (0/652)
ipc/	9.09% (28/308)	5.23% (278/5319)	2.82% (57/2018)	0.00% (0/149)
kernel/	32.13% (3093/9627)	23.56% (30751/130516)	15.24% (10261/67322)	5.93% (369/6226)
lib/	33.09% (716/2164)	24.59% (9217/37485)	18.42% (3500/19002)	9.73% (135/1388)
mm/	37.90% (1243/3280)	27.54% (15079/54753)	18.67% (5282/28298)	5.00% (137/2742)
net/	7.94% (1050/13219)	4.64% (12738/274249)	2.29% (3536/154676)	0.31% (46/14766)
security/	26.09% (359/1376)	14.29% (3359/23502)	10.57% (1265/11970)	1.22% (10/820)
sound/	3.94% (65/1648)	3.06% (812/26538)	1.68% (215/12804)	0.00% (0/1225)
Totals	17.92% (17030/95055)	11.64% (168997/1452209)	7.45% (53686/720962)	2.13% (1423/66672)

Generated by llvm-cov -- llvm version 20.0.0git

Summary page

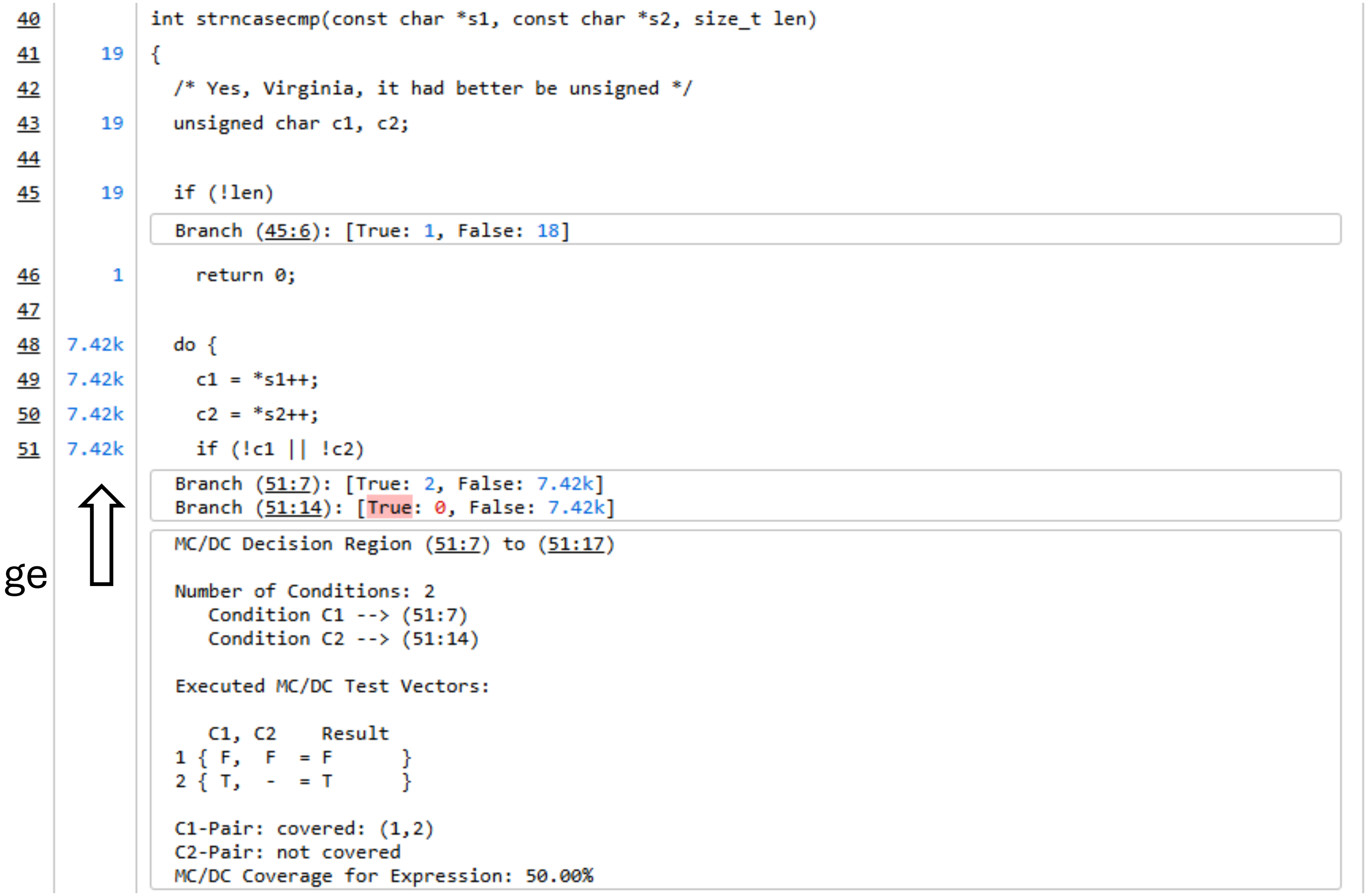
Example coverage report for KUnit tests



```
40          int strncasecmp(const char *s1, const char *s2, size_t len)
41      19  {
42            /* Yes, Virginia, it had better be unsigned */
43      19    unsigned char c1, c2;
44
45      19    if (!len)
            Branch (45:6): [True: 1, False: 18]

46       1      return 0;
47
48   7.42k    do {
49   7.42k      c1 = *s1++;
50   7.42k      c2 = *s2++;
51   7.42k      if (!c1 || !c2)
            Branch (51:7): [True: 2, False: 7.42k]
            Branch (51:14): [True: 0, False: 7.42k]

            MC/DC Decision Region (51:7) to (51:17)

            Number of Conditions: 2
               Condition C1 --> (51:7)
               Condition C2 --> (51:14)

            Executed MC/DC Test Vectors:

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

            C1-Pair: covered: (1,2)
            C2-Pair: not covered
            MC/DC Coverage for Expression: 50.00%
```

`lib/string.c` (tested by `string_kunit.c`)

Example coverage report for KUnit tests

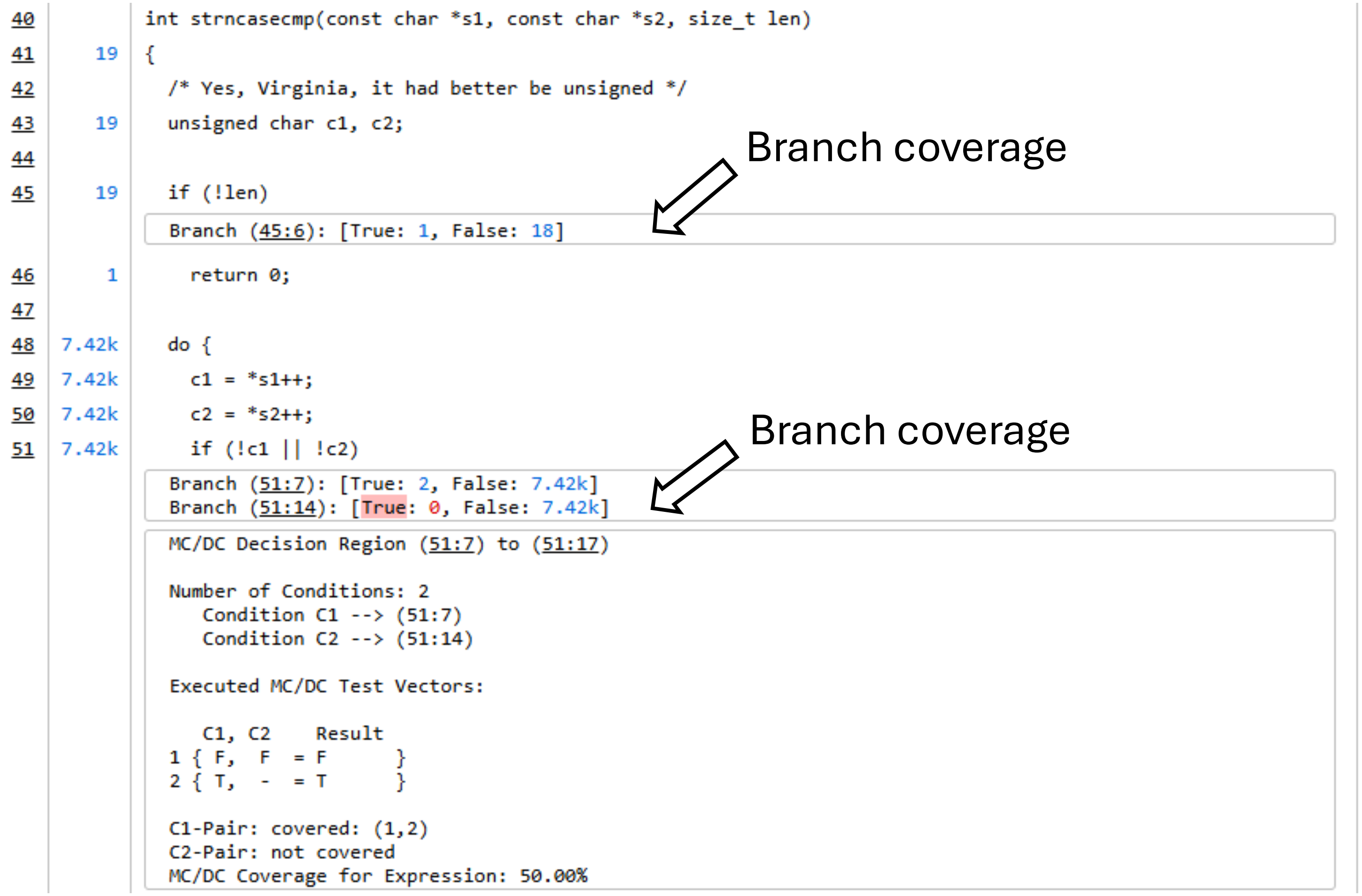


```
40          int strncasecmp(const char *s1, const char *s2, size_t len)
41      19  {
42            /* Yes, Virginia, it had better be unsigned */
43      19    unsigned char c1, c2;
44
45      19    if (!len)
              Branch (45:6): [True: 1, False: 18]
46       1      return 0;
47
48   7.42k    do {
49   7.42k      c1 = *s1++;
50   7.42k      c2 = *s2++;
51   7.42k      if (!c1 || !c2)
              Branch (51:7): [True: 2, False: 7.42k]
              Branch (51:14): [True: 0, False: 7.42k]

              MC/DC Decision Region (51:7) to (51:17)

              Number of Conditions: 2
                 Condition C1 --> (51:7)
                 Condition C2 --> (51:14)

              Executed MC/DC Test Vectors:

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

              C1-Pair: covered: (1,2)
              C2-Pair: not covered
              MC/DC Coverage for Expression: 50.00%
```

Line coverage

`lib/string.c` (tested by `string_kunit.c`)

Example coverage report for KUnit tests



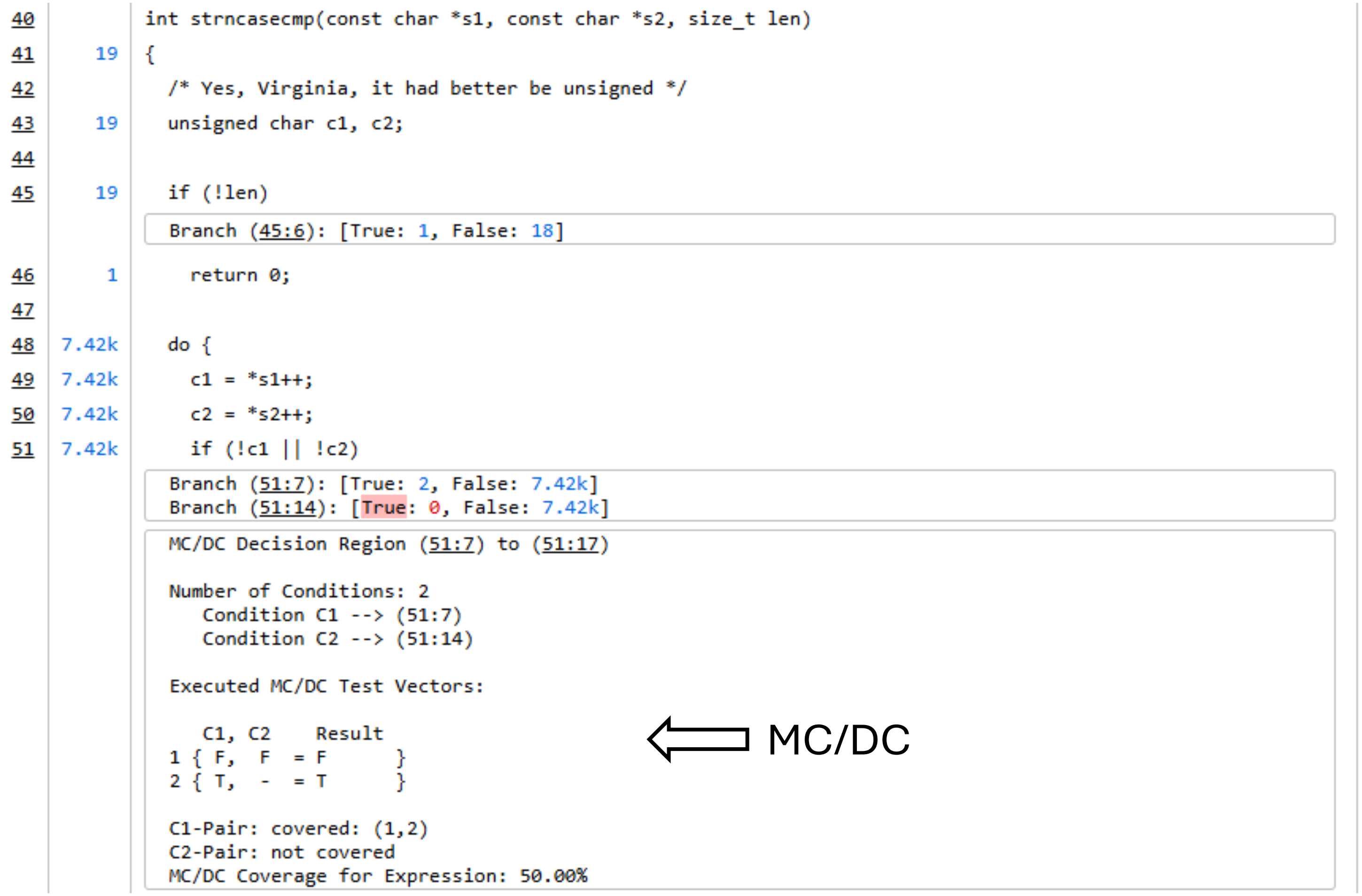
```
40          int strncasecmp(const char *s1, const char *s2, size_t len)
41      19  {
42            /* Yes, Virginia, it had better be unsigned */
43      19    unsigned char c1, c2;
44
45      19    if (!len)
              Branch (45:6): [True: 1, False: 18]

46       1      return 0;
47
48   7.42k    do {
49   7.42k      c1 = *s1++;
50   7.42k      c2 = *s2++;
51   7.42k      if (!c1 || !c2)
              Branch (51:7): [True: 2, False: 7.42k]
              Branch (51:14): [True: 0, False: 7.42k]
              MC/DC Decision Region (51:7) to (51:17)

              Number of Conditions: 2
                 Condition C1 --> (51:7)
                 Condition C2 --> (51:14)

              Executed MC/DC Test Vectors:

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

              C1-Pair: covered: (1,2)
              C2-Pair: not covered
              MC/DC Coverage for Expression: 50.00%
```

Branch coverage

Branch coverage

`lib/string.c` (tested by `string_kunit.c`)

Example coverage report for KUnit tests



```
40          int strncasecmp(const char *s1, const char *s2, size_t len)
41      19  {
42            /* Yes, Virginia, it had better be unsigned */
43      19    unsigned char c1, c2;
44
45      19    if (!len)
              Branch (45:6): [True: 1, False: 18]
46       1      return 0;
47
48   7.42k    do {
49   7.42k      c1 = *s1++;
50   7.42k      c2 = *s2++;
51   7.42k      if (!c1 || !c2)
              Branch (51:7): [True: 2, False: 7.42k]
              Branch (51:14): [True: 0, False: 7.42k]
              MC/DC Decision Region (51:7) to (51:17)

              Number of Conditions: 2
                 Condition C1 --> (51:7)
                 Condition C2 --> (51:14)

              Executed MC/DC Test Vectors:

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

              C1-Pair: covered: (1,2)
              C2-Pair: not covered
              MC/DC Coverage for Expression: 50.00%
```

MC/DC

`lib/string.c` (tested by `string_kunit.c`)

Known limitations

Known limitations

- Link-time warnings “call to `func()` leaves `.noinstr.text` section”

Known limitations

- Link-time warnings “call to `func()` leaves `.noinstr.text` section”
  - Possible solutions

Known limitations

- Link-time warnings “call to `func()` leaves `.noinstr.text` section”
    - Possible solutions
        - Modify LLVM inlining passes

Known limitations

- Link-time warnings "call to `func()` leaves `.noinstr.text` section"
    - Possible solutions
        - Modify LLVM inlining passes
        - Add `noinstr` attribute to `func()`

Known limitations

- Link-time warnings “call to `func()` leaves `.noinstr.text` section”
  - Possible solutions
    - Modify LLVM inlining passes
    - Add `noinstr` attribute to `func()`
- Spatial and temporal overhead

Known limitations

- Link-time warnings “call to `func()` leaves `.noinstr.text` section”
  - Possible solutions
    - Modify LLVM inlining passes
    - Add `noinstr` attribute to `func()`
- Spatial and temporal overhead
  - Potential alleviations
    - Separate information needed online and offline [3,4]

[3] https://discourse.llvm.org/t/instrprofiling-lightweight-instrumentation/59113
[4] https://discourse.llvm.org/t/rfc-add-binary-profile-correlation-to-not-load-profile-metadata-sections-into-memory-at-runtime/74565

Known limitations

- Link-time warnings "call to `func()` leaves `.noinstr.text` section"
  - Possible solutions
    - Modify LLVM inlining passes
    - Add `noinstr` attribute to `func()`
- Spatial and temporal overhead
  - Potential alleviations
    - Separate information needed online and offline [3,4]
    - Single byte counters [5]

[3] https://discourse.llvm.org/t/instrprofiling-lightweight-instrumentation/59113
[4] https://discourse.llvm.org/t/rfc-add-binary-profile-correlation-to-not-load-profile-metadata-sections-into-memory-at-runtime/74565
[5] https://discourse.llvm.org/t/rfc-single-byte-counters-for-source-based-code-coverage/75685

Known limitations (MC/DC-specific)

Known limitations (MC/DC-specific)

- Compile-time warnings for large decisions

Known limitations (MC/DC-specific)

- Compile-time warnings for large decisions
  - Maximum configurable through `Kbuild` option

Known limitations (MC/DC-specific)

- Compile-time warnings for large decisions
  - Maximum configurable through `Kbuild` option
- Large decisions incur prohibitive section size and can exceed `KERNEL_IMAGE_SIZE`

Known limitations (MC/DC-specific)

- Compile-time warnings for large decisions
  - Maximum configurable through `Kbuild` option
- Large decisions incur prohibitive section size and can exceed `KERNEL_IMAGE_SIZE`
  - Workarounds

Known limitations (MC/DC-specific)

- Compile-time warnings for large decisions
  - Maximum configurable through `Kbuild` option
- Large decisions incur prohibitive section size and can exceed `KERNEL_IMAGE_SIZE`
  - Workarounds
    - Measure on a per-subsystem basis

Known limitations (MC/DC-specific)

- Compile-time warnings for large decisions
  - Maximum configurable through `Kbuild` option
- Large decisions incur prohibitive section size and can exceed `KERNEL_IMAGE_SIZE`
  - Workarounds
    - Measure on a per-subsystem basis
    - Limit the number of conditions to be included

Known limitations (MC/DC-specific)

- Compile-time warnings for large decisions
  - Maximum configurable through `Kbuild` option
- Large decisions incur prohibitive section size and can exceed `KERNEL_IMAGE_SIZE`
  - Workarounds
    - Measure on a per-subsystem basis
    - Limit the number of conditions to be included
- Compile-time warnings for “split-nest” cases
  - E.g. `if (a && func(b && c))`

Known limitations (MC/DC-specific)

- Compile-time warnings for large decisions
  - Maximum configurable through `Kbuild` option
- Large decisions incur prohibitive section size and can exceed `KERNEL_IMAGE_SIZE`
  - Workarounds
    - Measure on a per-subsystem basis
    - Limit the number of conditions to be included
- Compile-time warnings for “split-nest” cases
  - E.g. `if (a && func(b && c))`
  - Can only be solved in LLVM upstream

Summary

Summary

- Precise **source-based coverage** produces more informative reports

Summary

- Precise **source-based coverage** produces more informative reports
- Propose **`kernel/llvm-cov/`** for unique insights into kernel codebase to testers, developers and certificate applicants

Summary

- Precise **source-based coverage** produces more informative reports
- Propose **`kernel/llvm-cov/`** for unique insights into kernel codebase to testers, developers and certificate applicants
  - It can coexist with `gcov` and `KCOV`

Summary

- Precise **source-based coverage** produces more informative reports
- Propose **`kernel/llvm-cov/`** for unique insights into kernel codebase to testers, developers and certificate applicants
  - It can coexist with gcov and KCOV
- Request for comments
  - LKML archive: https://lore.kernel.org/lkml/20240824230641.385839-1-wentaoz5@illinois.edu/
  - CI demo: https://github.com/xlab-uiuc/linux-mcdc

Backup slides for discussions

Feedback in LKML

- (Thomas Gleixner) Unifying `Makefile` variables `*COV_PROFILE`
  - Each `*cov` is implemented in different ways and separate lists are needed
- (Peter Zijlstra) `noinstr` attribute
  - It is correctly respected by the current toolchain

Old PGO debates

- https://lore.kernel.org/lkml/202106281231.E99B92BB13@keescook/
- Instrumentation (for precise coverage) vs. sampling (for profiling)
  - `perf` is not complete for coverage measurement, also hard to map back to source code

`syzkaller` discussion thread

- https://groups.google.com/g/syzkaller/c/JLX7ivDED5o
- Reuse `KCOV` interface
  - Needs compiler changes
- Measure coverage for specific processes (e.g. tests issued from user space) vs. overall execution

Future plan

- Support more architectures
- (Chuck Wolber) update LLVM intrinsics for more precise timing control

More backup slides

Backup: KCOV community is seeing similar problems

We find the current llvm coverage confusing as well (in the context of syzkaller/syzbot)

- Syzkaller mailing list discussion https://groups.google.com/g/syzkaller/c/JLX7ivDED5o
- One future work direction: per-test coverage

Backup: complete quote from LLVM docs

LLVM optimizations (such as inlining or CFG simplification) should have no impact on coverage report quality. This is due to the fact that the mapping from source regions to profile counters is immutable, and is generated before the LLVM optimizer kicks in. The optimizer can’t prove that profile counter instrumentation is safe to delete (because it’s not: it affects the profile the program emits), and so leaves it alone.

Note that this coverage feature does not rely on information that can degrade during the course of optimization, such as debug info line tables.

[1] https://clang.llvm.org/docs/SourceBasedCodeCoverage.html#impact-of-llvm-optimizations-on-coverage-reports

Backup: quote from `syzkaller` docs

Coverage is based on tracing coverage points inserted into the object code by the compiler. A coverage point generally refers to a basic block of code or a CFG edge (this depends on the compiler and instrumentation mode used during build, e.g. for Linux and clang the default mode is CFG edges, while for gcc the default mode is basic blocks). Note that coverage points are inserted by the compiler in the middle-end after a significant number of transformation and optimization passes. As the result coverage may poorly relate to the source code. For example, you may see a covered line after a non-covered line, or you may not see a coverage point where you would expect to see it, or vice versa (this may happen if the compiler splits basic blocks, or turns control flow constructs into conditional moves without control flow, etc). Assessing coverage is still generally very useful and allows to understand overall fuzzing progress, but treat it with a grain of salt.

https://github.com/google/syzkaller/blob/master/docs/coverage.md

Backup: steps to reproduce these `gcov` examples

- https://github.com/xlab-uiuc/linux-mcdc/issues/7

Backup: full-kernel instrumentation overhead

- Machine: PowerEdge R650 (kindly provided by CloudLab)
  - CPU: Two 36-core Intel Xeon Platinum 8360Y at 2.4GHz
  - RAM: 256GB ECC Memory (16x 16 GB 3200MHz DDR4)
- Clang: snapshot 20240917071600
  - For "apple-to-apple" comparison, here `gcov` is indeed Clang's `gcov` compatible mode, without MC/DC
- QEMU/KVM

	Build time	**`vmlinux` size**	**Boot time**	**Boot time w/ KUnit**
`noinstr`	53s	53M	2.25s	7.34s
`gcov`	1m10s	79M	2.40s	8.64s
`llvm-cov`	12m26s	1.3G	2.68s	9.80s