Code Coverage for GNU C Library

Eli Gurvitz
Functional Safety Architect,
Intel Mobileye

Ashutosh Pandey
CS Undergrad Student,
LFX Mentee

What is Code Coverage?

- Code coverage is the **percentage** of code which is covered by tests.

- Code coverage measurement simply determines which parts in a body of code have been executed through a test run, and which statements have not.

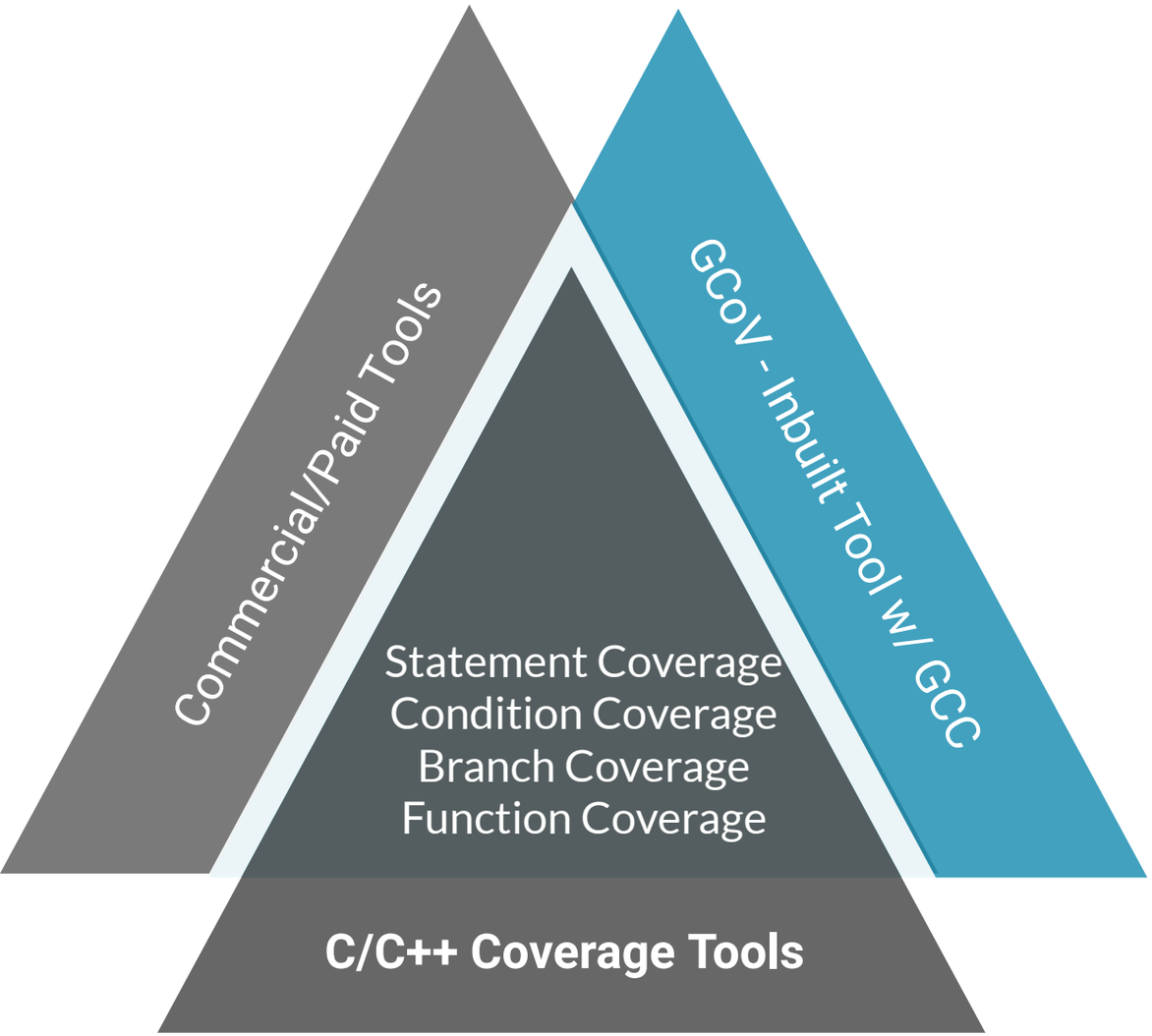
- In general, a code coverage system collects information about the running program and then combines that with source information to generate a report on the test suite's code coverage.

Why is it important?

- **Prove that the test suite is complete:** the tests visit all statements and branches

- **Prove that there isn't dead code:** if the test suite is complete but there is still unvisited code, then this is code that is not needed and can cause unpredictable behavior – normally it should be removed.

Code Coverage for C/C++



- **Commercial Tools:** Bullseye Coverage, Froglogic Coco, Testwell CTC++
- **Gcov:** Free/Open Source code coverage tool that comes with the GCC compiler. Widely used by many organizations.
- **Code Coverage Metrics:** Code coverage tools can show the %age (or fraction) of statements/functions/branches that are covered

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

Different Types Of Code Coverage

Statement Coverage

Is a metric that tells you whether the flow of control reached every executable statement of source code at least once.

Function coverage

This metric reports whether you invoked each function or procedure.

Branch coverage

Aims to ensure that each one of the possible branches from each decision point is executed at least once and thereby ensuring that all reachable code is executed.
eg: if statements and loops.

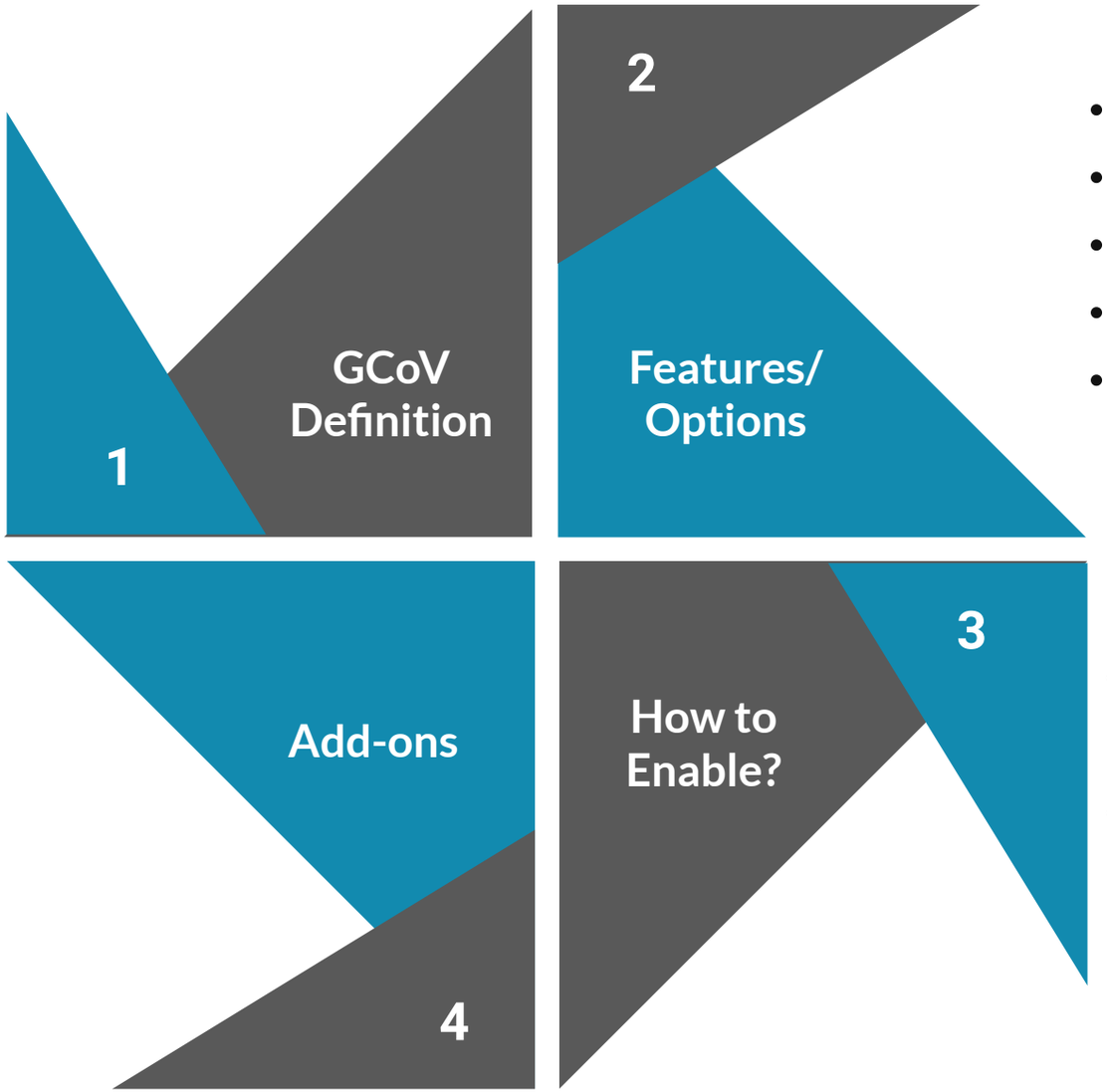
Condition Coverage

Show that each condition within a decision statement has been exercised.
eg: variables or sub expressions in a conditional statement.

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

gcov : A C/C++ Coverage Tool

- gcov is a source code coverage analysis tool.
- gcov generates exact counts of the number of times each statement in a program is executed.
- Lcov is a graphical front-end for gcov.
- Gcovr provides a utility for managing the use of gcov and generating summarized code coverage results.



- Branch Probabilities.
- Branch Counts.
- Function Summaries.
- All Blocks.
- Unconditional Branches.

- `$ gcc -fprofile-arcs -ftest-coverage filename.c`
- `$ gcov filename.c`

A Sample Program Covered Using Gcov

The following program, written in C, loops over the integers 1 to 9 and tests their divisibility with the modulus (%) operator.

```c
#include <stdio.h>

int
main (void)
{
  int i;

  for (i = 1; i < 10; i++)
    {
      if (i % 3 == 0)
        printf ("%d is divisible by 3\n", i);
      if (i % 11 == 0)
        printf ("%d is divisible by 11\n", i);
    }

  return 0;
}
```

Annotated Gcov File

```
ashutosh@ashu:~/glibc/test$ cat test.c
#include <stdio.h>

int
main (void)
{
  int i;

  for (i = 1; i < 10; i++)
    {
      if (i % 3 == 0)
        printf ("%d is divisible by 3\n", i);
      if (i % 11 == 0)
        printf ("%d is divisible by 11\n", i);
    }

  return 0;
}
ashutosh@ashu:~/glibc/test$ gcc -fprofile-arcs -ftest-coverage test.c
ashutosh@ashu:~/glibc/test$ ./a.out
3 is divisible by 3
6 is divisible by 3
9 is divisible by 3
ashutosh@ashu:~/glibc/test$ ls
a.out  test.c  test.gcda  test.gcno
ashutosh@ashu:~/glibc/test$ gcov -b -c test.c
File 'test.c'
Lines executed:85.71% of 7
Branches executed:100.00% of 6
Taken at least once:83.33% of 6
Calls executed:50.00% of 2
Creating 'test.c.gcov'
```

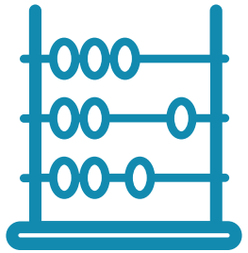
```
 1        -:    0:Source:test.c
 2        -:    0:Graph:test.gcno
 3        -:    0:Data:test.gcda
 4        -:    0:Runs:1
 5        -:    1:#include <stdio.h>
 6        -:    2:
 7        -:    3:int
 8 function main called 1 returned 100% blocks executed 89%
 9        1:    4:main (void)
10        -:    5:{
11        -:    6:  int i;
12        -:    7:
13       10:    8:  for (i = 1; i < 10; i++)
14 branch  0 taken 9
15 branch  1 taken 1 (fallthrough)
16        -:    9:    {
17        9:   10:      if (i % 3 == 0)
18 branch  0 taken 3 (fallthrough)
19 branch  1 taken 6
20        3:   11:        printf ("%d is divisible by 3\n", i);
21 call    0 returned 3
22        9:   12:      if (i % 11 == 0)
23 branch  0 taken 0 (fallthrough)
24 branch  1 taken 9
25    #####:   13:        printf ("%d is divisible by 11\n", i);
26 call    0 never executed
27        -:   14:    }
28        -:   15:
29        1:   16:  return 0;
30        -:   17:}
```

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

Gcov Data Files

```
ashutosh@ashu:~/test$ gcc -fprofile-arcs -ftest-coverage test.c
ashutosh@ashu:~/test$ ./a.out
3 is divisible by 3
6 is divisible by 3
9 is divisible by 3
ashutosh@ashu:~/test$ ls
a.out  test.c  test.gcda  test.gcno
```

- **.gcno notes file** is generated when the source file is compiled with the GCC -ftest-coverage option. It contains information to reconstruct the basic block graphs and assign source line numbers to blocks.



- **.gcda count data file** is generated when a program containing object files built with the GCC -fprofile-arcs option is executed. It contains arc transition counts, value profile counts, and some summary information.

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

How Does Gcov Work Underneath?

- For our given sample program, we have a few questions:
- What changes when we add `-fprofile-arcs -ftest-coverage` as a compiler option?
- Is the flow of the program altered in any way?
- How is the coverage data counted and written to the gcda data files?

```c
#include <stdio.h>

int
main (void)
{
  int i;

  for (i = 1; i < 10; i++)
    {
      if (i % 3 == 0)
        printf ("%d is divisible by 3\n", i);
      if (i % 11 == 0)
        printf ("%d is divisible by 11\n", i);
    }

  return 0;
}
```

Compiling Without Coverage

```
ashutosh@ashu:~/glibc/test$ gcc -E test.c -o test.i
ashutosh@ashu:~/glibc/test$ gcc -S test.i
ashutosh@ashu:~/glibc/test$ as -o test.o test.s
ashutosh@ashu:~/glibc/test$ gcc -o test test.o
ashutosh@ashu:~/glibc/test$ nm test.o
                 U _GLOBAL_OFFSET_TABLE_
0000000000000000 T main
                 U printf
ashutosh@ashu:~/glibc/test$
```

```
gcc -o test test.c
nm test.o                                   //View the symbols in the test.o file
```

```
                 U _GLOBAL_OFFSET_TABLE_
0000000000000000 T main
                 U printf
```

Compiling With Coverage Option Enabled

- From the results of the nm command, we can see that the `test.o` file has 4 more symbols:
- **gcov_init**, **gcov_exit**, **gcov_main**, gcov_merge_add.
- Now we will compare and analyse the assembly code before and after coverage, and then discuss the analysis of the inserted instrumentation code.

```
ashutosh@ashu:~/glibc/test$ gcc -E test.c -o test.i
ashutosh@ashu:~/glibc/test$ gcc -fprofile-arcs -ftest-coverage -S test.i
ashutosh@ashu:~/glibc/test$ as -o test.o test.s
ashutosh@ashu:~/glibc/test$ gcc -o test test.o
/usr/bin/ld: test.o: in function `_sub_I_00100_0':
test.c:(.text+0x131): undefined reference to `__gcov_init'
/usr/bin/ld: test.o: in function `_sub_D_00100_1':
test.c:(.text+0x140): undefined reference to `__gcov_exit'
/usr/bin/ld: test.o:(.data.rel+0x20): undefined reference to `__gcov_merge_add'
collect2: error: ld returned 1 exit status
ashutosh@ashu:~/glibc/test$ nm test.o
0000000000000000 b __gcov0.main
                 U __gcov_exit
                 U __gcov_init
0000000000000000 d __gcov_.main
                 U __gcov_merge_add
                 U _GLOBAL_OFFSET_TABLE_
0000000000000000 T main
                 U printf
0000000000000137 t _sub_D_00100_1
0000000000000121 t _sub_I_00100_0
ashutosh@ashu:~/glibc/test$
```

Analysing The Assembly Before Coverage

```c
#include <stdio.h>

int
main (void)
{
  int i;

  for (i = 1; i < 10; i++)
    {
      if (i % 3 == 0)
        printf ("%d is divisible by 3\n", i);
      if (i % 11 == 0)
        printf ("%d is divisible by 11\n", i);
    }

  return 0;
}
```

Here , the program statements on top correspond to the assembly output on the right.

```
.LC0:
        .string "%d is divisible by 3\n"
.LC1:
        .string "%d is divisible by 11\n"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 1
        jmp     .L2
.L5:
        mov     edx, DWORD PTR [rbp-4]
        movsx   rax, edx
        imul    rax, rax, 1431655766
        shr     rax, 32
        mov     ecx, edx
        sar     ecx, 31
        sub     eax, ecx
        mov     ecx, eax
        add     ecx, ecx
        add     ecx, eax
        mov     eax, edx
        sub     eax, ecx
        test    eax, eax
        jne     .L3
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
.L3:
        mov     ecx, DWORD PTR [rbp-4]
        movsx   rax, ecx
        imul    rax, rax, 780903145
        shr     rax, 32
        sar     eax
        mov     esi, ecx
        sar     esi, 31
        sub     eax, esi
        mov     edx, eax
        mov     eax, edx
        sal     eax, 2
        add     eax, edx
        add     eax, eax
        add     eax, edx
        mov     edx, ecx
        sub     edx, eax
        test    edx, edx
        jne     .L4
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    printf
.L4:
        add     DWORD PTR [rbp-4], 1
.L2:
        cmp     DWORD PTR [rbp-4], 9
        jle     .L5
        mov     eax, 0
        leave
        ret
```

Assembly After Coverage

```
.LC0:
        .string "%d is divisible by 3\n"
.LC1:
        .string "%d is divisible by 11\n"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     rax, QWORD PTR __gcov0.main[rip]
        add     rax, 1
        mov     QWORD PTR __gcov0.main[rip], rax
        mov     DWORD PTR [rbp-4], 1
        jmp     .L2
.L5:
        mov     rax, QWORD PTR __gcov0.main[rip+40]
        add     rax, 1
        mov     QWORD PTR __gcov0.main[rip+40], rax
        mov     edx, DWORD PTR [rbp-4]
        movsx   rax, edx
        imul    rax, rax, 1431655766
        shr     rax, 32
        mov     ecx, edx
        sar     ecx, 31
        sub     eax, ecx
        mov     ecx, eax
        add     ecx, ecx
        add     ecx, eax
        mov     eax, edx
        sub     eax, ecx
        test    eax, eax
        jne     .L3
        mov     rax, QWORD PTR __gcov0.main[rip+8]
        add     rax, 1
        mov     QWORD PTR __gcov0.main[rip+8], rax
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     rax, QWORD PTR __gcov0.main[rip+16]
        add     rax, 1
        mov     QWORD PTR __gcov0.main[rip+16], rax
.L3:
        mov     ecx, DWORD PTR [rbp-4]
        movsx   rax, ecx
        imul    rax, rax, 780903145
        shr     rax, 32
        sar     eax
        mov     esi, ecx
        sar     esi, 31
        sub     eax, esi
        mov     edx, eax
        mov     eax, edx
        sal     eax, 2
        add     eax, edx
        add     eax, eax
        add     eax, edx
        mov     edx, ecx
        sub     edx, eax
        test    edx, edx
        jne     .L4
        mov     rax, QWORD PTR __gcov0.main[rip+24]
        add     rax, 1
        mov     QWORD PTR __gcov0.main[rip+24], rax
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    printf
.L4:
        add     DWORD PTR [rbp-4], 1
        mov     rax, QWORD PTR __gcov0.main[rip+32]
        add     rax, 1
        mov     QWORD PTR __gcov0.main[rip+32], rax
.L2:
        cmp     DWORD PTR [rbp-4], 9
        jle     .L5
        mov     eax, 0
        leave
        ret
__gcov_.main:
        .quad   .LPBX0
        .long   108032747
        .long   2063057141
        .long   560455345
        .zero   4
        .long   6
        .zero   4
        .quad   __gcov0.main
.LC2:
        .string "/home/ce/./output.gcda"
.LPBX0:
        .long   1110454826
        .zero   4
        .quad   0
        .long   -166487736
        .zero   4
        .quad   .LC2
        .quad   __gcov_merge_add
        .quad   0
        .quad   0
        .quad   0
        .quad   0
        .quad   0
        .quad   0
        .quad   0
        .quad   0
        .long   1
        .zero   4
        .quad   .LPBX1
```

Code Stub Analysis

- A total of 7 stub codes have been inserted, out of which the first 6 are easy to understand.
- It is a counter, and as long as the relevant code is executed every time, the counter is increased by 1.
- It can be seen from the stub below that the function to be completed by this code is actually to increase this counter by 1 , but what is this counter?

```
movq    __gcov0.main(%rip), %rax
addq    $1, %rax
movq    %rax, __gcov0.main(%rip)
```

- It is a long integer of 8 bytes composed of .LPBX0 and __gcov_main + 6 . The first 6 stubs are actually for a static array with 6 long integer elements. The size of each counter is 8 bytes.

```
__gcov_.main:
        .quad   .LPBX0
        .long   108032747
        .long   -345659544
        .long   560455345
        .zero   4
        .long   6
        .zero   4
        .quad   __gcov0.main
        .section        .rodata
        .align 8
```

- From the section properties of __gcov_main: we can see that the array should be RODATA (read only data).
- After the code is executed, the value of the array records the number of times the stub code is executed, that is, the number of times the subsequent code block is executed as seen here.

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

```
81  __gcov_.main:
82          .quad   .LPBX0
83          .long   108032747
84          .long   2063057141
85          .long   560455345
86          .zero   4
87          .long   6
88          .zero   4
89          .quad   __gcov0.main
90  .LC2:
91          .string "/home/ce/./output.gcda"
92  .LPBX0:
93          .long   1110454826
94          .zero   4
95          .quad   0
96          .long   -164426054
97          .zero   4
98          .quad   .LC2
99          .quad   __gcov_merge_add
100         .quad   0
101         .quad   0
102         .quad   0
103         .quad   0
104         .quad   0
105         .quad   0
106         .quad   0
107         .long   1
108         .zero   4
109         .quad   .LPBX1
110 .LPBX1:
111         .quad   __gcov_.main
112 _sub_I_00100_0:
113         pushq   %rbp
114         movq    %rsp, %rbp
115         movl    $.LPBX0, %edi
116         call    __gcov_init
117         popq    %rbp
118         ret
```

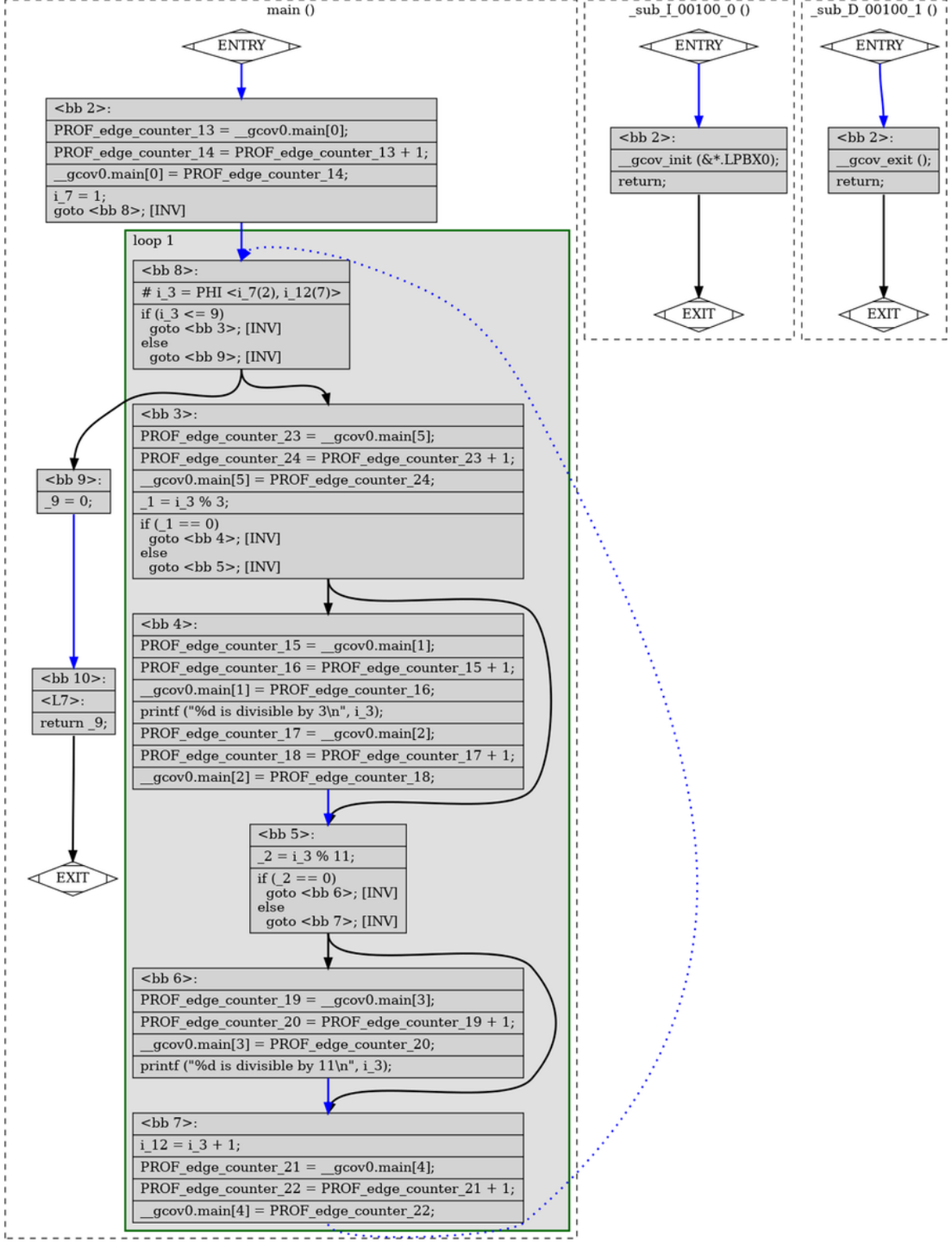
Analysis of The Constructor

- The last stub inserted into the program is always the same, regardless of the contents of the program.
- It can be seen that this is a function named **__gcov_main** . The main purpose of this function is to call the **__gcov_init function**. The calling parameter is the **.LPBX0** structure.
- Using the objdump command on the executable file, we can see the symbols.

```
Disassembly of section .data.rel.local:

0000000000000000 <__gcov_.main>:
       ...
   8:   eb 72                   jmp    7c <__gcov_.main+0x7c>
   a:   70 06                   jo     12 <__gcov_.main+0x12>
   c:   68 a7 65 eb b1          pushq  $0xffffffffb1eb65a7
  11:   de 67 21                fisubs 0x21(%rdi)
  14:   00 00                   add    %al,(%rax)
  16:   00 00                   add    %al,(%rax)
  18:   06                      (bad)
       ...
```

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS



After All The Compiler Passes

- To represent the control flow visually, we find the control flow graph (CFG).
- $ gcc -fdump-tree-all-graph test.c -o test
- If we generate the CFG after the compiler passes, we get the following CFG.
- `$ dot -Tpng test.c.231t.optimized.dot -o cfg0pt.png`
- We can see gcov main being called for each block, this corresponds to the stubs we saw earlier.

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

The gcov_init Function

```
void
__gcov_init (struct gcov_info *info)
{
  if (!info->version || !info->n_functions)
    return;
  if (gcov_version (info, info->version, 0))
    {
      if (!__gcov_root.list)
        {
          /* Add to master list and at exit function.  */
          if (gcov_version (NULL, __gcov_master.version, "<master>"))
            {
              __gcov_root.next = __gcov_master.root;
              if (__gcov_master.root)
                __gcov_master.root->prev = &__gcov_root;
              __gcov_master.root = &__gcov_root;
            }
        }

      info->next = __gcov_root.list;
      __gcov_root.list = info;
    }
}
```

- The **__gcov_init** function is defined in `libgcov-driver.c`
- From this we can draw two conclusions:
- **.LPBX0** structure is **gcov_info** structure, the two are the same.
- **__gcov_init** features the .LPBX0 structure, i.e. gcov_info structure, strung together a list, the list pointer is **gcov_list** .

Data Structure Analysis

```
/* Information about a single object file.  */
struct gcov_info
{
  gcov_unsigned_t version;      /* expected version number */
  struct gcov_info *next;       /* link to next, used by libgcov */

  gcov_unsigned_t stamp;        /* uniquifying time stamp */
  const char *filename;         /* output file name */

  gcov_merge_fn merge[GCOV_COUNTERS];  /* merge functions (null for
                                          unused) */

  unsigned n_functions;         /* number of functions */

#ifndef IN_GCOV_TOOL
  const struct gcov_fn_info *const *functions; /* pointer to pointers
                                                  to function information  */
#else
  struct gcov_fn_info **functions;
  struct gcov_summary summary;
#endif /* !IN_GCOV_TOOL */
};
```

- `gcov.c` describes how the cycles are counted for profiling purposes.
- Hawick's cycle detection algorithm is used to find all the simple paths in a subgraph.
- The functions for creating solution flow graphs are taken from `profile.c` in GCC, which is used for profiling.
- The gcno and gcda file formats are documented in `gcov-io.h`.

ISO 26262

ISO 26262, titled "Road vehicles – Functional safety", is an **international standard for functional safety of electrical and/or electronic systems** in serial production road vehicles.

Why Is It Required?

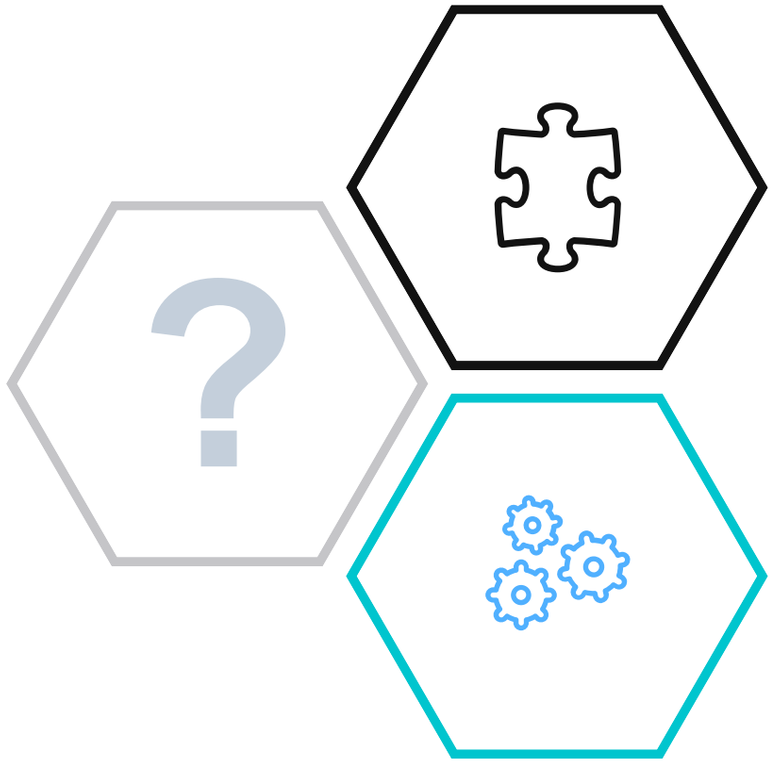
Functional Safety standards, such as ISO 26262:2016 part 6 section 9.4.5, require the measurement of structural (code) coverage in order to evaluate the completeness of requirements and tests. When GLibC is used in a safety-related application, and there are safety requirements allocated to GLibC, then code coverage analysis must be provided for GLibC.

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

Challenges While Building With Code Coverage

- Configure failed when we added coverage flags as arguments to the configure command
- There is a circular dependency between the gcov instrumentation and glibc
- The instrumentation calls gcov_merge_add
- gcov_merge_add calls fopen and other file operations
- Shared libraries could not be built
- Static library build flow depended on files generated during shared library build

What Are The Errors We Face?



Undefined References

Several libraries, like **libgcov.a** and **lgcc** are not linked, and causes the build to fail.

Shared Libraries

While the build process goes further along by disabling all the shared libraries, several libraries are not covered.

Inconsistencies with GLibC versions

Initially we developed the solution for GLibC 2.32, but later it worked for 2.31 and earlier versions too.

Is There A Way To Make The Build Pass?

- By carefully selecting configure time options, and by noting down compiler errors and manually including the right libraries, the build can be made to pass.
- However this is very arbitrary and dependent on the program being built.
- An example is given below:

```
$ ../configure --prefix=/home/ashutosh/glibc/install-gcov --disable-shared CFLAGS="-
fprofile-arcs -ftest-coverage-O2"

$ make //make fails at some point because libgcov.a was not found, we add it manually.

$ gcc -nostdlib -nostartfiles -static -o
/home/ashutosh/glibc/build_gcov/iconv/iconvconfig/home/ashutosh/glibc/build_gcov/csu/crt1.
o /home/ashutosh/glibc/build_gcov/csu/crti.o `gcc--print-file-name=crtbeginT.o`
/home/ashutosh/glibc/build_gcov/iconv/iconvconfig.o/home/ashutosh/glibc/build_gcov/iconv/s
trtab.o
/home/ashutosh/glibc/build_gcov/iconv/xmalloc.o/home/ashutosh/glibc/build_gcov/iconv/hash-
string.o -Wl,--start-group /home/ashutosh/glibc/build_gcov/libc.a-lgcc -lgcc_eh -Wl,--end-
group `gcc --print-file-name=crtend.o` /home/ashutosh/glibc/build_gcov/csu/crtn.o-lgcov

//build passes successfully
```

01 A Patch to GLibC

02 A Patch to GCC (libgcc)

03 Adding a configure option to GLibC

The Patch

What are the different ways to approach the patch?

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

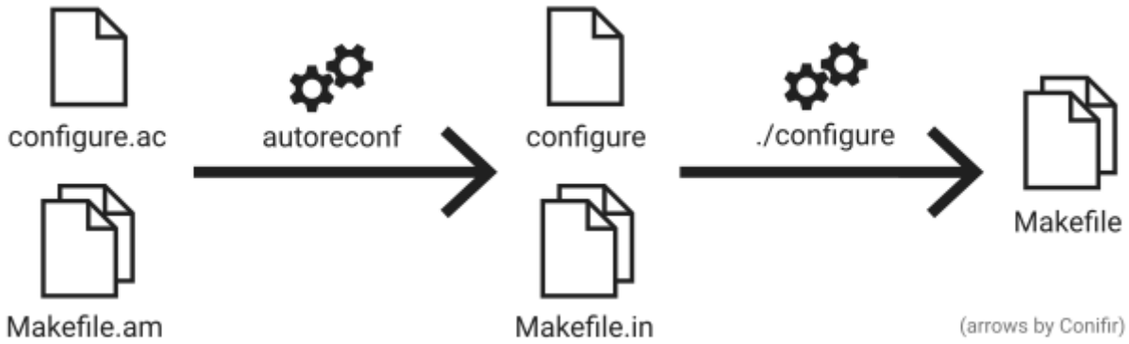
Why Add A Configure Option For Coverage?

- **It is self contained:** altering the libraries to make them link properly may have side effects on other parts of the library.
- **It is optional:** unless the option is enabled explicitly, coverage enabled libraries are not built.
- **The solution is compact:** With a few changes to the files involved in the build process such as `Makeconfig`, `configure.ac` and `configure`, we can make the coverage work effortlessly.
- **Fail early:** In case something is not configured properly to allow coverage enabled shared libraries, a message at configure time is better than an obscure error part way through the build.

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

What Files Need To Be Changed?

- **Makeconfig**: to provide information about the libraries to be linked.
- **configure.ac**: to add the extra configure option.
- **configure**: To check for shared libraries during configure time, and display messages if configuration process failed.
- **Documentation**: In the INSTALL, TexInfo Manual, and News section.



Makeconfig

- On **line 613** in `Makeconfig`:

```
# support for code coverage with gcov

ifeq (yes,$(build-gcov))
+cflags += -fprofile-arcs -ftest-coverage -O2
link-libc-static = -Wl,--start-group -lgcov $(common-objpfx)libc.a $(static-gnulib) -Wl,--
end-group
link-libc-static-tests = -Wl,--start-group -lgcov $(common-objpfx)libc.a $(static-gnulib-
tests) -Wl,--end-group
endif
```

configure.ac & config.make.in

- On **line 178** in `configure.ac`:

```
AC_ARG_ENABLE([gcov],
        AC_HELP_STRING([--enable-gcov],
         [build with instrumentation for gcov code coverage library @<:@default=no@:>@]),
           [gcov=$enableval],
           [gcov=no])
AC_SUBST(gcov)
```

On **line 95** in `config.make.in`:

```
build-gcov = @gcov@
```

configure

- On **line 686** in `configure`:

```
gcov
```

- On **line 770** in configure:

```
enable_gcov
```

- On **line 1437** in configure:

```
--enable-gcov           build with instrumentation for gcov code coverage
                          [default=no]
```

- On **line 3380** in configure:

```
# Check whether --enable-gcov was given.
if test "${enable_gcov+set}" = set; then :
  enableval=$enable_gcov; gcov=$enableval
else
  gcov=no
fi
```

Documentation

- On **line 114** in INSTALL:

```
'--enable-gcov'
    Enable building glibc with instrumentation for code coverage. For
    individual files '--coverage' can be used, but this fails
    when it is used as an option with '../configure' command to build glibc.
    To successfully build glibc with coverage, you must first build and install glibc
    with shared libraries enabled. This will allow some files such as
    'libc-modules.h' to be generated which are needed for code coverage to
    work.
    The command '../configure --disable-shared --enable-gcov --without-selinux
    --disable-nscd --prefix=/path/to/install'
    will build glibc with code coverage. The command to make is 'make CXX=', without
    which you will get a error 'cannot find -lgcc_s'.
```

How To Build With Coverage?

01

Normal configure & Build

02

Special configure command

03

make CXX= & make install

- First we need to configure and build glibc normally. All these steps should be done with a glibc on an **alternate system root**, or installed with a different prefix.
- This step is needed so that the necessary shared libraries, such as `libc-modules.h` are generated correctly.
- Next, we'll use our configure command:

```
../configure -gcov --disable-shared --disable-nscd --without-selinux prefix=/usr
```

- Followed by `make CXX=`
- Finally, with `make install`, we are done and all the .gcno files are generated.
- To verify that we have the .gcno files:

```
find . -name \*.gcno -print
```

Sample Output

```
ashutosh@ashu:~/glibc/build$ find . -name \*.gcno
./libio/putwc.gcno
./libio/clearerr.gcno
./libio/iovswscanf.gcno
./libio/putc.gcno
./libio/putwc_u.gcno
./libio/iofgetws_u.gcno
./libio/fileops.gcno
./libio/fileno.gcno
./libio/oldfileops.gcno
./libio/getchar.gcno
./libio/oldiofgetpos64.gcno
./libio/iofgetpos64.gcno
./libio/fseeko.gcno
./libio/getc.gcno
./libio/iogetline.gcno
./libio/iopopen.gcno
./libio/iofopen.gcno
./libio/__fwritable.gcno
./libio/iofclose.gcno
./libio/iofflush.gcno
./libio/fseeko64.gcno
./libio/iofread_u.gcno
./libio/oldstdfiles.gcno
./libio/iopadn.gcno
./libio/putwchar_u.gcno
./libio/fcloseall.gcno
./libio/__fbufsize.gcno
./libio/iofflush_u.gcno
./libio/oldiofopen.gcno
./libio/libc_fatal.gcno
./libio/getwc_u.gcno
./libio/fwprintf.gcno
./libio/wprintf.gcno
./libio/ftello64.gcno
./libio/putchar_u.gcno
posix/confstr.gcno
posix/getaddrinfo.gcno
posix/execle.gcno
posix/spawn_faction_destroy.gcno
posix/setresuid.gcno
posix/globfree.gcno
posix/spawni.gcno
posix/spawn_faction_addopen.gcno
posix/getopt.gcno
posix/glob.gcno
posix/spawnattr_getsigmask.gcno
posix/execvp.gcno
posix/spawnattr_setschedparam.gcno
posix/spawnattr_getdefault.gcno
posix/pathconf.gcno
posix/environ.gcno
posix/sched_setaffinity.gcno
posix/sysconf.gcno
posix/spawnattr_getflags.gcno
posix/pread64.gcno
posix/pwrite64.gcno
posix/streams-compat.gcno
posix/spawnattr_setdefault.gcno
posix/nanosleep.gcno
posix/spawn_faction_addchdir.gcno
posix/glob-lstat-compat.gcno
posix/spawn_faction_adddup2.gcno
posix/pause.gcno
posix/glob64-lstat-compat.gcno
posix/wait4.gcno
posix/setpgrp.gcno
posix/sched_cpucount.gcno
posix/spawnattr_destroy.gcno
posix/execvpe.gcno
posix/spawnattr_getschedparam.gcno
posix/oldglob.gcno
./shadow/getspnam.gcno
./shadow/putspent.gcno
./shadow/lckpwdf.gcno
./shadow/getspent_r.gcno
./shadow/getspent.gcno
./shadow/fgetspent_r.gcno
./shadow/getspnam_r.gcno
./wcsmbs/wcsrchr.gcno
./wcsmbs/wcrtomb.gcno
./wcsmbs/wcsnrtombs.gcno
./wcsmbs/wcsnlen-c.gcno
./wcsmbs/wcstod_l.gcno
./wcsmbs/wcsrtombs.gcno
./wcsmbs/wcstold_nan.gcno
./wcsmbs/mbsrtowcs.gcno
./wcsmbs/wcstod.gcno
./wcsmbs/wmemcmp.gcno
./wcsmbs/wcscoll_l.gcno
./wcsmbs/wcsncpy.gcno
./wcsmbs/isoc99_swscanf.gcno
./wcsmbs/wcstoull_l.gcno
./wcsmbs/wcscspn.gcno
./wcsmbs/wcspbrk.gcno
./wcsmbs/wcstol_l.gcno
./wcsmbs/isoc99_fwscanf.gcno
./wcsmbs/wcschrnul.gcno
./wcsmbs/wcstold.gcno
./wcsmbs/wcstof128_l.gcno
./wcsmbs/wcwidth.gcno
./wcsmbs/mbrtoc32.gcno
./wcsmbs/wmempcpy.gcno
./wcsmbs/isoc99_vwscanf.gcno
./wcsmbs/wcscasecmp.gcno
./wcsmbs/wcswidth.gcno
./wcsmbs/wcpcpy.gcno
./wcsmbs/wcsdup.gcno
./elf/rtld-dl-addr-obj.gcno
./elf/dl-scope.gcno
./elf/dl-tls.gcno
./elf/rtld.gcno
./elf/dl-execstack.gcno
./elf/dl-call-libc-early-init.gcn
./elf/dl-load.gcno
./elf/dl-conflict.gcno
./elf/dl-cet.gcno
./elf/dl-object.gcno
./elf/dl-profile.gcno
./elf/dl-error-minimal.gcno
./elf/dl-open.gcno
./elf/dl-misc.gcno
./elf/dl-sort-maps.gcno
./elf/dl-iteratephdr.gcno
./elf/dl-fini.gcno
./elf/dl-sysdep.gcno
./elf/dl-version.gcno
./elf/dl-sbrk.gcno
./elf/dl-getcwd.gcno
./elf/dl-reloc.gcno
./elf/dl-init.gcno
./elf/dl-lookup-direct.gcno
./elf/dl-get-cpu-features.gcno
./elf/dl-origin.gcno
./elf/dl-tunables.gcno
./elf/dl-close.gcno
./elf/dl-error.gcno
./elf/dl-deps.gcno
./elf/dl-brk.gcno
./elf/dl-runtime.gcno
./elf/dl-opendir.gcno
./elf/dl-sym.gcno
./elf/dl-openat64.gcno
./elf/libc_early_init.gcno
```

Setting Up GLibC For Coverage

- # Get glibc-2.32 and untar it to <root> directory
- cd <root>
- mkdir build-gcov
- cd build-gcov
- ../configure -prefix=/usr
- make -j
- Export DESTDIR=<directory to install the library files>
- make install DESTDIR=${DESTDIR}
- cd ..
- # Get the **glibc_coverage.patch** file
- git apply glibc_coverage.patch file
- cd build-gcov
- ../configure -prefix=/usr -enable-gcov -disable-shared -disable-nscd -without-selinux
- # Note that the dashes in the previous line are 2 dashes each
- make CXX=
- make install DESTDIR=${DESTDIR}
- cd ../..
- mkdir test
- cd test

Building The Test

```
# Build the test with a command like the below. Change it first to:
# Change the sysroot to the folder where you installed the glib library (DESTDIR)
# change the file name marked in bold:

SYSROOT=/localdrive/users/egurvitz/wip/glibc/install-gcov
gcc -static -fprofile-arcs -ftest-coverage \
  -L${SYSROOT}/usr/lib64 \
  -I${SYSROOT}/usr/include \
  --sysroot=${SYSROOT} \
  -Wl,-rpath=${SYSROOT}/lib64 \
  -Wl,--dynamic-linker=${SYSROOT}/lib64/ld-2.32.so\
  -g -o malloctest malloctest.c ${SYSROOT}/usr/lib64/libc.a /usr/lib/gcc/x86_64-linux-gnu/7/libgcov.a
# Run the executable
cd ../glibc-2.32/malloc  # (malloc is an example. Go to the location of the source file)
Gcov -f -b ../build-gcov/malloc/malloc.c
# The previous line will give the gcov stats and also create the file malloc.c.gcov in the current folder.
```

Summary

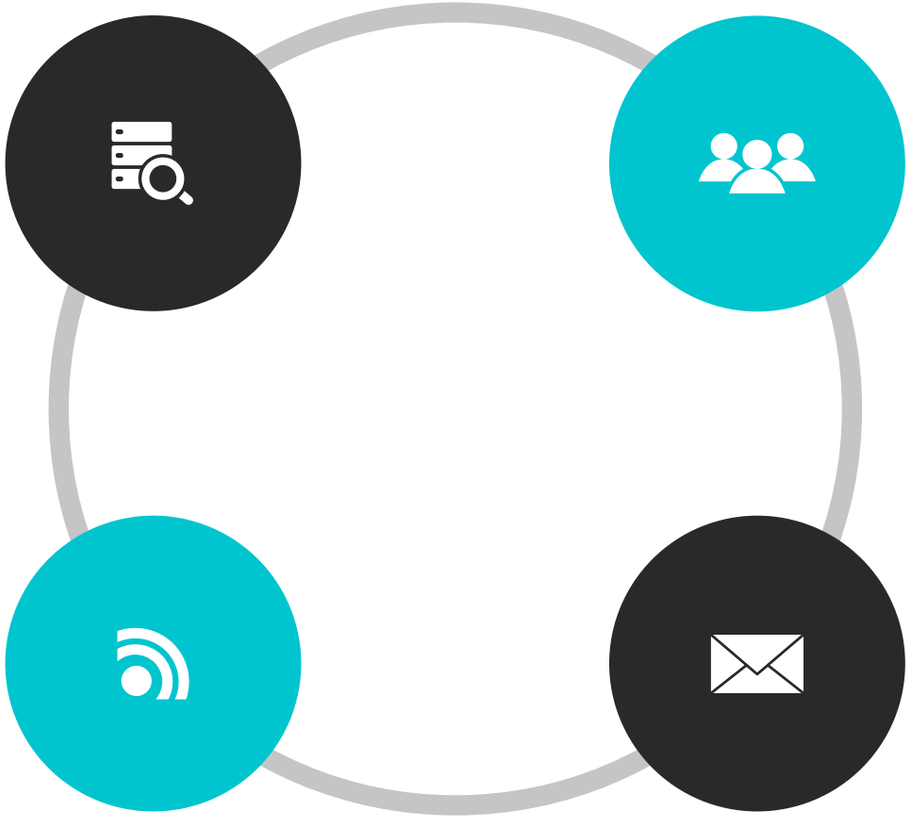
In this session we covered:

Code Coverage

and its importance.

Gcov

its uses and shortcomings.



Patch

a fix for code coverage of shared libraries in glibc.

Applications

how to build your own application with coverage enabled.

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

THANK YOU
FOR LISTENING

Do you have any questions?

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS