The ‘better’ Makefile

Firstly, the prerequisite: knowing what a Makefile is and how to generate a basic one (for our typical C on Linux use case). If hazy, do refer this superb article:
What is a Makefile and how does it work?, S Patil, Red Hat, Aug 2018

Where is this so-called ‘better Makefile?
- userspace ‘C’ : Makefile
- kernel module : Makefile

Why is it *better*? Simpy because it allows you to perform valuable checks and analysis very easily; *stuff you might otherwise forget/ignore/put off for ever!*

Requirements
You will need these installed on your build host:
- (cross) compiler (GCC/clang)
  - the Makefile checks; if clang is installed, it uses it in preference to GCC (except for coverage)
- make
- indent
- checkpatch.pl
- static analysis tools
  - sparse
  - flawfinder
  - cppcheck
- dynamic analysis tools
  - valgrind
  - gcov, lcov, lcov_gen.sh (code coverage analysis)
- tar

Examples of stuff it can catch:

The examples below are when using this Makefile’s targets on a few system programming apps on 5.4 Linux.

Code styling

Linux kernel coding style forced via indent(1) and checked via the kernel source tree *scripts/checkpatch.pl* Perl script!

```
...
--- applying Linux kernel code-style checking with checkpatch.pl ---
```

```
../../checkpatch.pl -f --no-tree --max-line-length=95 *.[ch]
No typos will be found - file
'/mnt/big/kaiwan/Dropbox/DG_Work_Dropbox/github_kaiwan_repos/trg/L1_sysprg_trg/
spelling.txt': No such file or directory
No structs that should be const will be found - file
'/mnt/big/kaiwan/Dropbox/DG_Work_Dropbox/github_kaiwan_repos/trg/L1_sysprg_trg/
const_structs.checkpatch': No such file or directory
WARNING: Missing or malformed SPDX-License-Identifier tag in line 1
#1: FILE: envp.c:1:
+/* envp.c:

WARNING: braces {} are not necessary for single statement blocks
#22: FILE: envp.c:22:
+     for (i = 0; environ[i] != NULL; i++) {
+          envp[i] = environ[i];
+     }

ERROR: trailing whitespace
#28: FILE: envp.c:28:
+^I * Note the use of the strncpy (as opposed to strcpy) : this is for $

WARNING: Block comments use * on subsequent lines
#42: FILE: envp.c:42:
+     /*
+        gcc gives: envp.c:42: warning: missing sentinel in function call

ERROR: trailing whitespace
#45: FILE: envp.c:45:
+^I   "The list of arguments must be terminated by a NULL pointer, and, $

...


WARNING: struct  should normally be const
#78: FILE: mysh.c:78:
+static void show_resusage(struct rusage *rusage)

...

ERROR: do not use assignment in if condition
#205: FILE: mysh.c:205:
+               if ((cpid = wait3(&stat, WUNTRACED, rusage)) == -1)

...

ERROR: code indent should use tabs where possible
#32: FILE: libpk.h:32:
+                        x=(inner_index%2)*((22/7)%3); \$
...

WARNING: please, no spaces at the start of a line
#29: FILE: sched_pthrd_rtprio.c:29:
```

```
+   setting sched policy to SCHED_FIFO and RT priority to %ld in 2 seconds..\
n", getpid(), (long)msg);$
...
WARNING: Prefer using '"%s...", __func__' to using 'main', this function's
name, in a string
#107: FILE: sched_pthrd_rtprio.c:107:
+    printf("main thread (%d): now creating realtime pthread p2..\n",
...
```

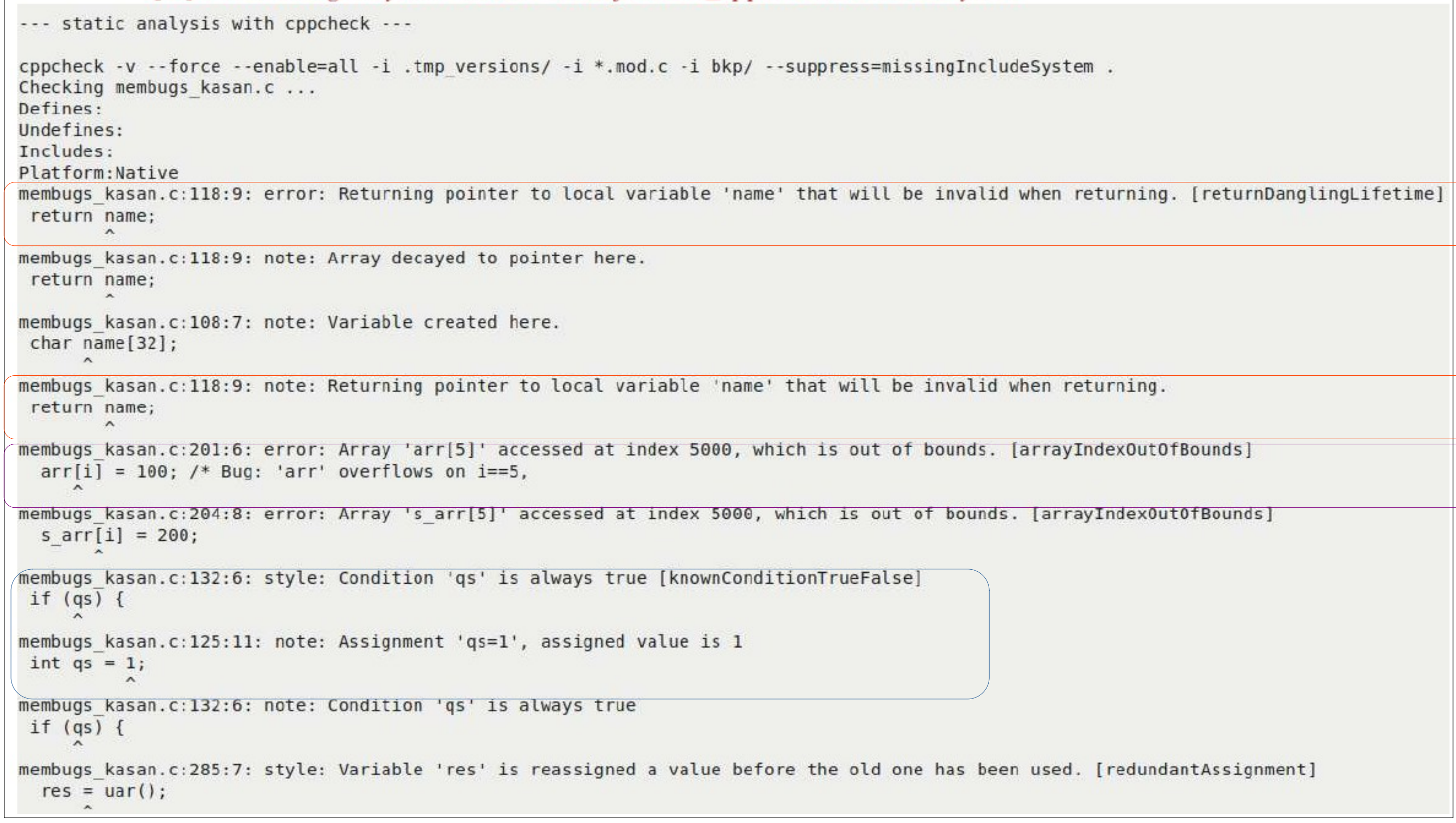
Static Analysis

Interestingly, NEITHER valgrind nor ASAN catches the dangerous UAR – Use After Return / Use After Scope – bug!

Static analysis – here, cppcheck – catches the UAR !

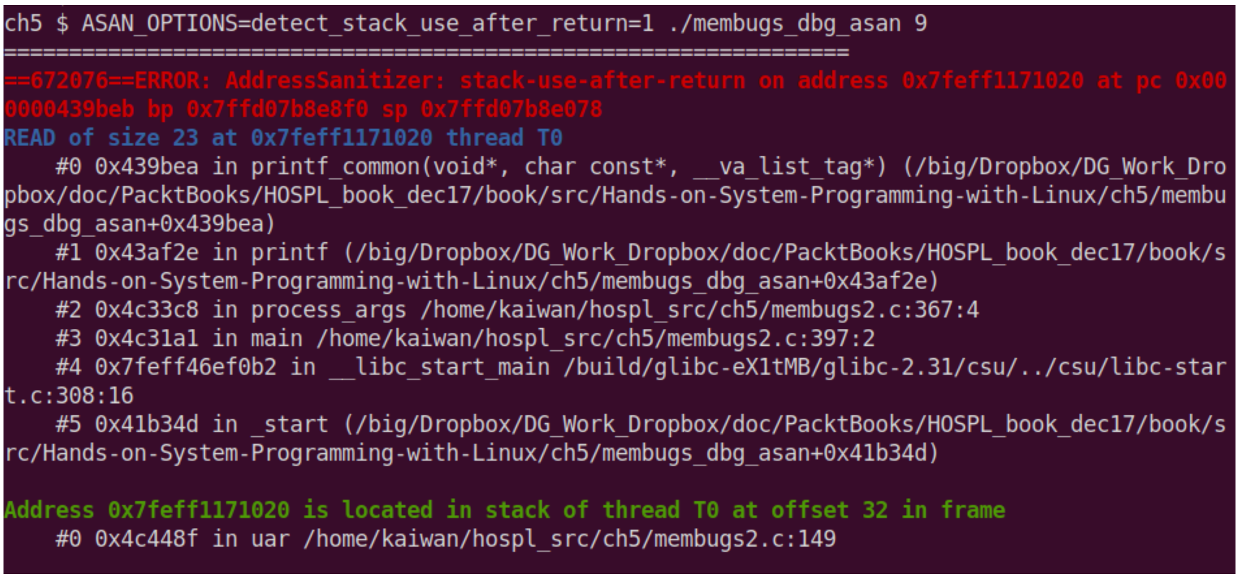
Also caught by cppcheck static analysis

**[C] : UAR caught by our 'better' Makefile's sa_cppcheck static analysis !*



```
--- static analysis with cppcheck ---

cppcheck -v --force --enable=all -i .tmp_versions/ -i *.mod.c -i bkp/ --suppress=missingIncludeSystem .
Checking membugs_kasan.c ...
Defines:
Undefines:
Includes:
Platform:Native
membugs_kasan.c:118:9: error: Returning pointer to local variable 'name' that will be invalid when returning. [returnDanglingLifetime]
 return name;
        ^
membugs_kasan.c:118:9: note: Array decayed to pointer here.
 return name;
        ^
membugs_kasan.c:108:7: note: Variable created here.
 char name[32];
      ^
membugs_kasan.c:118:9: note: Returning pointer to local variable 'name' that will be invalid when returning.
 return name;
        ^
membugs_kasan.c:201:6: error: Array 'arr[5]' accessed at index 5000, which is out of bounds. [arrayIndexOutOfBounds]
  arr[i] = 100; /* Bug: 'arr' overflows on i==5,
     ^
membugs_kasan.c:204:8: error: Array 's_arr[5]' accessed at index 5000, which is out of bounds. [arrayIndexOutOfBounds]
  s_arr[i] = 200;
       ^
membugs_kasan.c:132:6: style: Condition 'qs' is always true [knownConditionTrueFalse]
 if (qs) {
     ^
membugs_kasan.c:125:11: note: Assignment 'qs=1', assigned value is 1
 int qs = 1;
          ^
membugs_kasan.c:132:6: note: Condition 'qs' is always true
 if (qs) {
     ^
membugs_kasan.c:285:7: style: Variable 'res' is reassigned a value before the old one has been used. [redundantAssignment]
  res = uar();
      ^
```

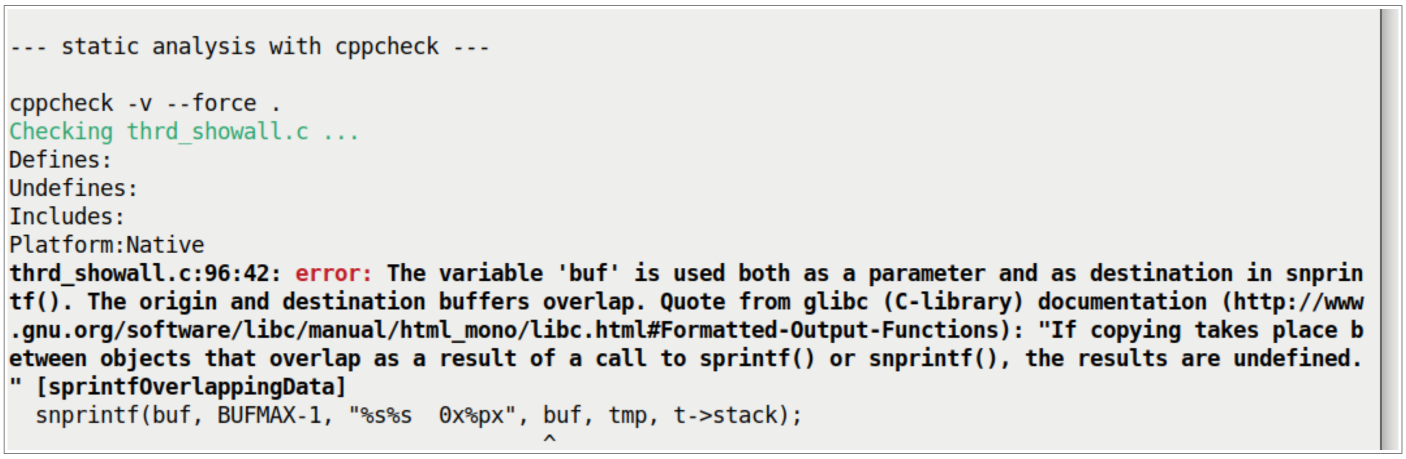
[NOTE/FYI- ASAN can catch UAR if `detect_stack_use_after_return` is set to 1:



```
ch5 $ ASAN_OPTIONS=detect_stack_use_after_return=1 ./membugs_dbg_asan 9
=================================================================
==672076==ERROR: AddressSanitizer: stack-use-after-return on address 0x7feff1171020 at pc 0x00
0000439beb bp 0x7ffd07b8e8f0 sp 0x7ffd07b8e078
READ of size 23 at 0x7feff1171020 thread T0
    #0 0x439bea in printf_common(void*, char const*, __va_list_tag*) (/big/Dropbox/DG_Work_Dro
pbox/doc/PacktBooks/HOSPL_book_dec17/book/src/Hands-on-System-Programming-with-Linux/ch5/membu
gs_dbg_asan+0x439bea)
    #1 0x43af2e in printf (/big/Dropbox/DG_Work_Dropbox/doc/PacktBooks/HOSPL_book_dec17/book/s
rc/Hands-on-System-Programming-with-Linux/ch5/membugs_dbg_asan+0x43af2e)
    #2 0x4c33c8 in process_args /home/kaiwan/hospl_src/ch5/membugs2.c:367:4
    #3 0x4c31a1 in main /home/kaiwan/hospl_src/ch5/membugs2.c:397:2
    #4 0x7feff46ef0b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-star
t.c:308:16
    #5 0x41b34d in _start (/big/Dropbox/DG_Work_Dropbox/doc/PacktBooks/HOSPL_book_dec17/book/s
rc/Hands-on-System-Programming-with-Linux/ch5/membugs_dbg_asan+0x41b34d)

Address 0x7feff1171020 is located in stack of thread T0 at offset 32 in frame
    #0 0x4c448f in uar /home/kaiwan/hospl_src/ch5/membugs2.c:149
```

```
...]
```

cppcheck also caught this potential bug once:



```
--- static analysis with cppcheck ---

cppcheck -v --force .
Checking thrd_showall.c ...
Defines:
Undefines:
Includes:
Platform:Native
thrd_showall.c:96:42: error: The variable 'buf' is used both as a parameter and as destination in snprin
tf(). The origin and destination buffers overlap. Quote from glibc (C-library) documentation (http://www
.gnu.org/software/libc/manual/html_mono/libc.html#Formatted-Output-Functions): "If copying takes place b
etween objects that overlap as a result of a call to sprintf() or snprintf(), the results are undefined.
" [sprintfOverlappingData]
  snprintf(buf, BUFMAX-1, "%s%s  0x%px", buf, tmp, t->stack);
                                         ^
```

Flawfinder – a simple static analyser biased towards security checking

```
...
flawfinder --neverignore --context *.[ch]
Flawfinder version 2.0.10, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
Examining fork2c.c

FINAL RESULTS:

envp.c:48:  [4] (shell) execle:
  This causes a new program to execute and is difficult to use safely
  (CWE-78). try using a library call that implements the same functionality
  if available.
      if (execle("/usr/bin/printenv", "printenv", (char *)0, envp) < 0) {
```

```
envp.c:19:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
     static char *envp[MAXENV]; /* Pointers to environment */

[...]

fork2c.c:66:  [2] (integer) atoi:
  Unless checked, the resulting number can exceed the expected range
  (CWE-190). If source untrusted, check both minimum and maximum, even if the
  input had no minus sign (large numbers can roll over into negative number;
  consider saving to an unsigned value if that is intended).
          c1_slptm = atoi(argv[1]);

<< you should instead use the:
userspace: strtol     strtold    strtoll    strtoq     strtoul    strtoull ...
kernel   : kstrtoint  kstrtoint_from_user  kstrtol  kstrtol_from_user  kstrtoll
            kstrtoll_from_user ...
>>


...

mysh_simple.c:31:  [4] (format) vfprintf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification.
     vfprintf(stderr, fmt, ap);
...

mysh_simple.c:50:  [1] (buffer) strlen:
  Does not handle strings that are not \0-terminated; if given one it may
  perform an over-read (it could cause a crash if unprotected) (CWE-126).
          cmd[strlen(cmd) - 1] = '\0';    /* remove trailing \n */
...
```

From: https://burkhardstubert.substack.com/p/episode-29-better-built-by-burkhard
...
Expert Talk with Adam Tornhill: Code Refactoring
A static analysis tool spits out 15,000 issues for your code base and tells you that it takes roughly 10 person years to fix them all. This information is **useless**. Adam Tornhill makes the information useful by ranking the issues by how often the code around the issue changes. *The more often the code changes the more important the issue is.*

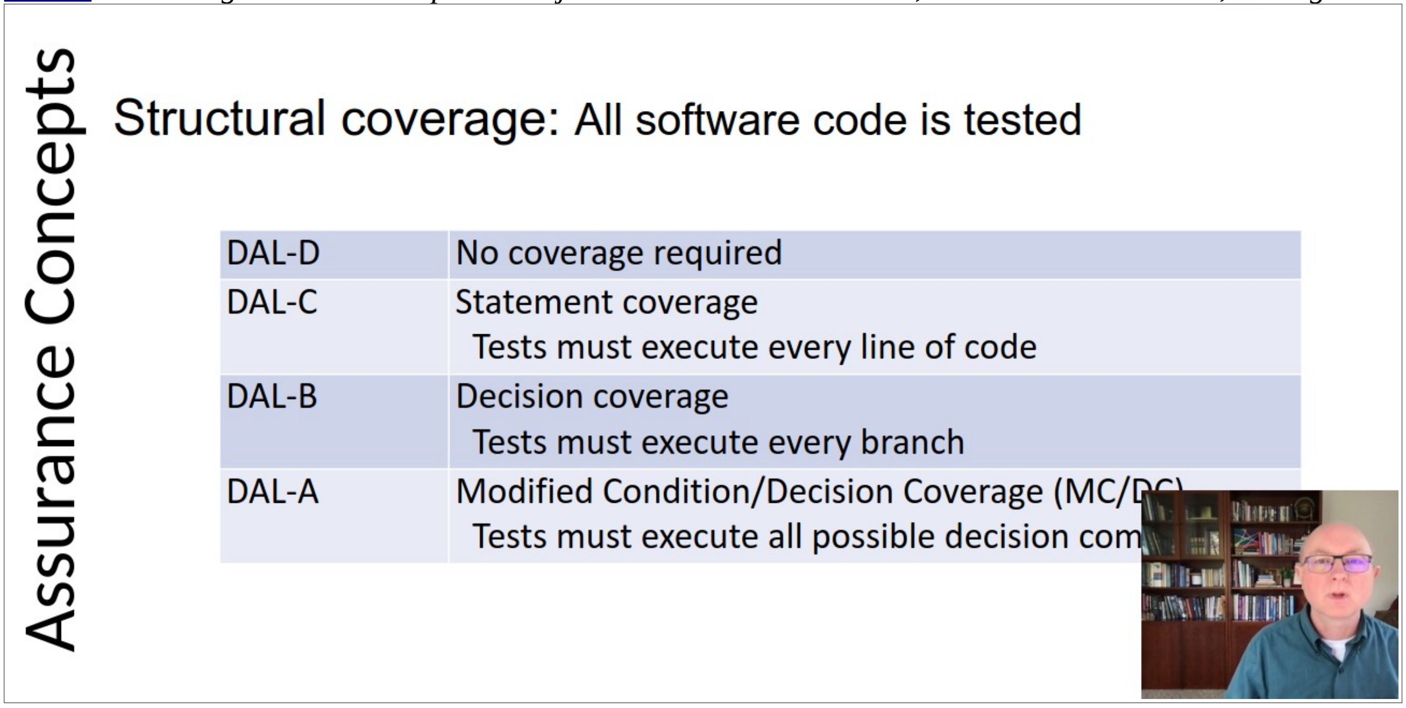
Adam has built the tool CodeScene that combines information from multiple sources like static analysis tools, GitHub, Jenkins and JIRA to come up with the ranking. You may also be interested in Adam's book Software Design X-Rays: Fix Technical Debt with Behavorial Code Analysis.
...

Code Coverage Analysis

Why?
Dynamic analysers check code as they run it.. BUT they *cannot possibly* check code that they don't run! Hence, we have to ensure that most/all lines of code are actually run, by running test cases (postive & negative), fuzzers, etc, and prove – via code coverage analysis – that all lines did actually get exercised.

The amount of code coverage depends on the safety assurance standard that needs to be met for the project.. For example, in avionics, it's obviously very high.

Source *: Debating Linux in Aerospace - Objections and Paths Forward, Dr. Steven Vanderleest, Boeing*



Assurance Concepts

Structural coverage: All software code is tested

DAL-D	No coverage required
DAL-C	Statement coverage Tests must execute every line of code
DAL-B	Decision coverage Tests must execute every branch
DAL-A	Modified Condition/Decision Coverage (MC/DC) Tests must execute all possible decision com

* DAL-A: Modified Condition/Decision Coverage (MC/DC). Tests must execute all possible decision combinations.

Code Coverage analysis with gcov and lcov

Steps:

1. Build for code coverage analysis:
gcc test.c -o test_gcov -fprofile-arcs -ftest-coverage -lgcov -Wall

2. Execute it:
./test_gcov

3. Capture the coverage tracefile via lcov:
lcov --base-directory . --directory . --capture --output-file test.info

4. Run genhtml to generate the HTML report:
genhtml -o html_report/ test.info

5. Open html_report/index.html in a web browser to see the coverage report.

Our ‘better’ Makefile and lcov_gen.sh coverage wrapper script automate the collection and reporting process. Try it!

LCOV! - is a graphical front-end to gcov.

To make use of it:
1. Install the lcov package and relatd dependencies...
```
sudo apt install lcov libcapture-tiny-perl libdatetime-perl
```

2. Get this convenience script:
https://github.com/kaiwan/usefulsnips/blob/master/lcov_gen.sh

3. Use the ‘better’ Makefile
https://github.com/kaiwan/L1_sysprg_trg/blob/master/0_Makefile_template/killer/Makefile

<<
Possible issue on Ubuntu (Apr 2024):
[SRU] "Can't locate lcovutil.pm" after updating to 2.0-1

The current – buggy – version is:

```
$ dpkg -l|grep lcov
ii  lcov                        2.0-1                        all        Summarise Code coverage
information from GCOV
$ lcov
Can't locate lcovutil.pm in @INC (you may need to install the lcovutil module) (@INC contains:
/usr/local/lib/lcov /etc/perl /usr/local/lib/x86_64-linux-gnu/perl/5.36.0 /usr/local/share/perl/5.36.0
/usr/lib/x86_64-linux-gnu/perl5/5.36 /usr/share/perl5 /usr/lib/x86_64-linux-gnu/perl-base
/usr/lib/x86_64-linux-gnu/perl/5.36 /usr/share/perl/5.36 /usr/local/lib/site_perl) at /usr/bin/lcov line
102.
BEGIN failed--compilation aborted at /usr/bin/lcov line 102.
$
```

Need to upgrade to the very latest version as the current one's buggy... to do so, we have to add the mantic-proposed repository to the repo list:

```
# vi /etc/apt/sources.list
…
# for lcov
deb http://gb.archive.ubuntu.com/ubuntu/ mantic-proposed mantic-proposed multiverse main universe restricted
#
```

After this, remove and install it via the specific new repo:

```
# apt remove lcov
…
# apt install -t mantic-proposed lcov
…
$ dpkg -l|grep lcov
ii  lcov                    2.0-1ubuntu0.2                 all        Summarise Code
coverage information from GCOV
```

Now works!
>>

Example (here):

```
$ cat test2.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

// QuickPrint !
#define QP printf(" In %s:%s():%d\n", __FILE__, __func__, __LINE__);

void foo(void)
{
	QP;
}
void bar(void)
{
	QP;
}
void oof(void)
{
	QP;
}
int main(int argc, char **argv)
{
	if (argc != 2) {
		fprintf(stderr, "Usage: %s case# [1-3]\n", argv[0]);
		exit(1);
	}
```

```
      switch (atoi(argv[1])) {
      case 1:
            printf("case 1\n");
            foo();
            break;
      case 2:
            printf("case 2\n");
            bar();
            break;
      case 3:
            printf("case 3\n");
            oof();
            break;
      default:
            printf("invalid case\n");
            exit(1);
      }
      exit(0);
}
$
```

I Manually performing a code coverage run

```
$ ls
Makefile  test2.c  test.c
$ gcc test2.c -o test2_gcov -fprofile-arcs -ftest-coverage -lgcov -Wall
$ ls -l test2_gcov
-rwxrwxr-x 1 kaiwan kaiwan 27696 Apr  7 17:39 test2_gcov*
$ ./test2_gcov
Usage: ./test2_gcov case# [1-3]
$ ./test2_gcov 1
case 1
 In test2.c:foo():10
$
$ lcov  --base-directory . --directory . --capture --output-file test2.info
Capturing coverage data from .
geninfo cmd: '/usr/local/bin/geninfo . --output-filename test2.info --base-directory .
--memory 0 --branch-coverage'
Found gcov version: 13.2.0
Using intermediate gcov format
Writing temporary data to /tmp/geninfo_datOU2z
Scanning . for .gcda files ...
Found 1 data files in .
Processing ./test2_gcov-test2.gcda
Finished .info-file creation
$
$ ls -l test2.info                    <-- the coverage tracefile
-rw-rw-r-- 1 kaiwan kaiwan 570 Apr  7 17:40 test2.info
$ genhtml -o html_report/ test2.info
Found 1 entries.
Found common filename prefix
"/big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug"
Generating output.
Processing file coverage_test/test2.c
  lines=30 hit=13 functions=4 hit=2 branches=6 hit=3
Overall coverage rate:
  lines......: 43.3% (13 of 30 lines)
  functions......: 50.0% (2 of 4 functions)
  branches......: 50.0% (3 of 6 branches)
$
$ ls -l
total 72
drwxrwxr-x 3 kaiwan kaiwan  4096 Apr  7 17:40 html_report/
-rw-rw-r-- 1 kaiwan kaiwan 18256 Apr  7 16:51 Makefile
-rw-rw-r-- 1 kaiwan kaiwan   580 Apr  5 11:53 test2.c
-rwxrwxr-x 1 kaiwan kaiwan 27696 Apr  7 17:39 test2_gcov*
-rw-rw-r-- 1 kaiwan kaiwan   268 Apr  7 17:39 test2_gcov-test2.gcda
-rw-rw-r-- 1 kaiwan kaiwan  1950 Apr  7 17:39 test2_gcov-test2.gcno
-rw-rw-r-- 1 kaiwan kaiwan   570 Apr  7 17:40 test2.info
-rw-rw-r-- 1 kaiwan kaiwan   192 Apr  5 11:53 test.c
$ ls -l html_report/
total 88
...
drwxrwxr-x 2 kaiwan kaiwan  4096 Apr  7 17:40 coverage_test/
-rw-rw-r-- 1 kaiwan kaiwan   141 Apr  7 17:40 emerald.png
...
-rw-rw-r-- 1 kaiwan kaiwan  5896 Apr  7 17:40 index.html
-rw-rw-r-- 1 kaiwan kaiwan  5889 Apr  7 17:40 index-sort-b.html
-rw-rw-r-- 1 kaiwan kaiwan  5889 Apr  7 17:40 index-sort-f.html
-rw-rw-r-- 1 kaiwan kaiwan  5889 Apr  7 17:40 index-sort-l.html
...
$
```

II Automating it via our 'better' Makefile and lcov_gen.sh script

```
$ make help
Compiler set to clang
=== Makefile Help : additional targets available ===
...
...
--- code coverage ---
 covg       : run the gcov+lcov code coverage tooling on the source (generates
html output!). NOTE: this target requires our lcov_gen.sh wrapper script installed
(location: https://github.com/kaiwan/usefulsnips/blob/master/lcov_gen.sh)
...
$
```

First, ensure the command-line parameters (if any) to the PUT (Program Under Test), are set in the Makefile:

```
$ cat Makefile
...
FNAME_C := test2
#--- CHECK: manually add params as required
# Populate any required cmdline arguments to the process here:
CMDLINE_ARGS="1"
...
```

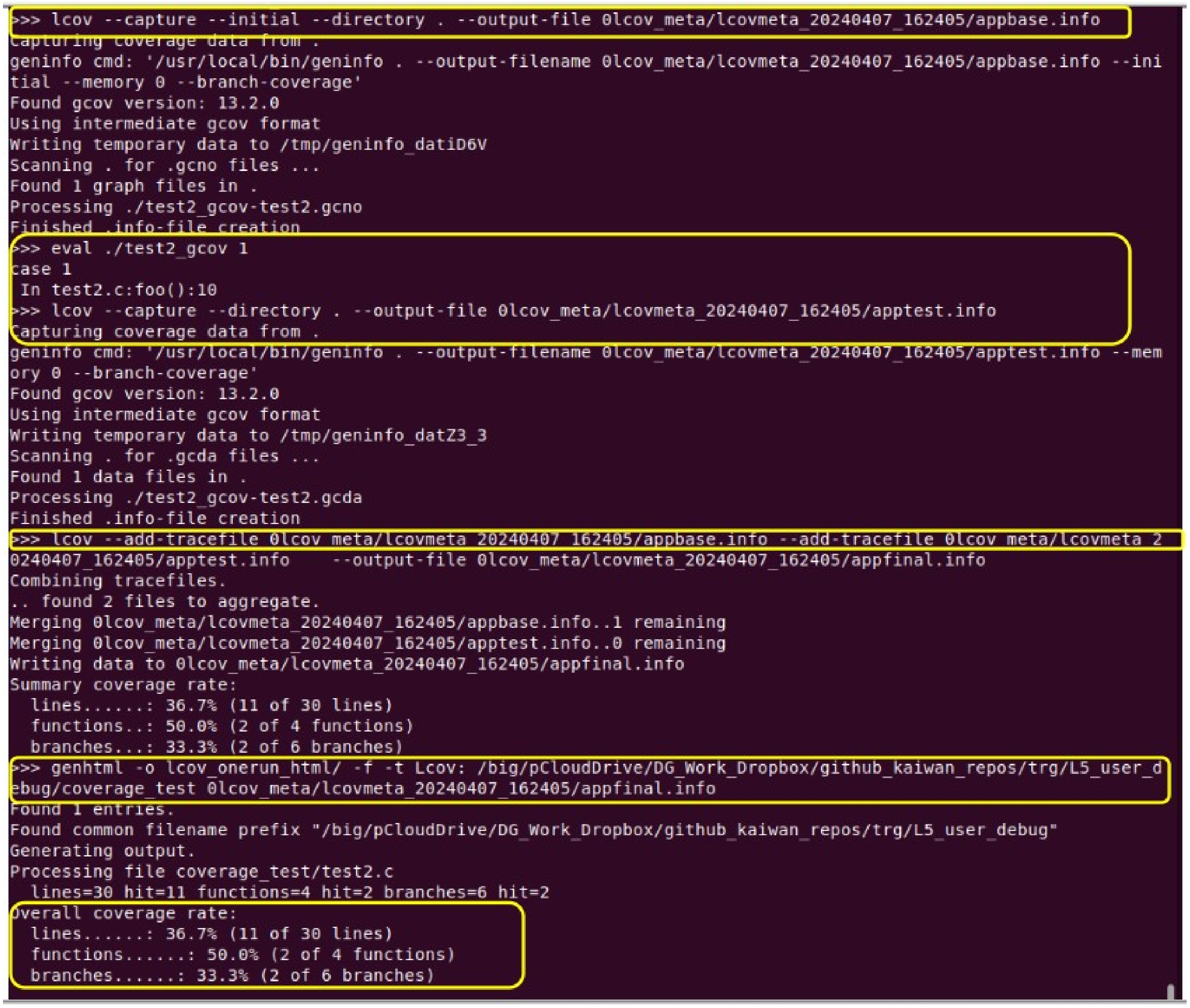
Good.
The '**covg**' target has the Makefile:

1. Detect the presence of the ~/.lcovrc lcov startup file
2. Via sed, edits it to ensure that locv/genhtml branch coverage are enabled
3. Performs the 'clean', followed by building the coverage-enabled binary executable (passing the `-fprofile-arcs -ftest-coverage -lgcov` options among other debug options)

    1. Uses GCC, as gcov/lcov seem to require it (and not clang)
4. Checks for the CMDLINE_ARGS variable being initialized and emits a warning if not
5. Invokes our wrapper lcov_gen.sh coverage script, which actually runs lcov and genhtml to generate both the one-time and (by default) merged coverage reports!

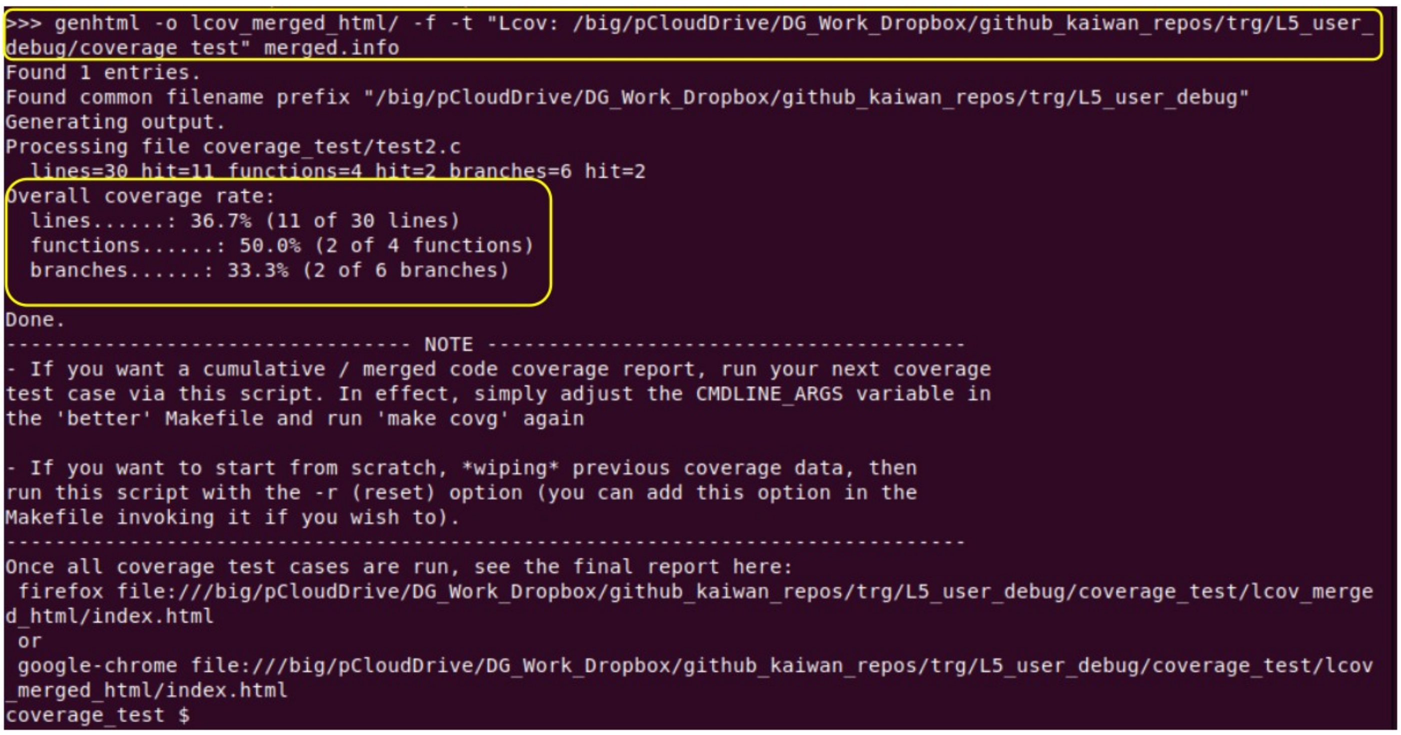
Run #1 of 3: with CMDLINE_ARGS="1"

```
$ make covg
Compiler set to clang

=== Code coverage (funcs/lines/branches) with gcov+lcov ===
...
...
> Forcing compiler to GCC for coverage, as gcov/lcov seem to require it
gcc -g -ggdb -gdwarf-4 -O0 -Wall -Wextra -DDEBUG -fno-omit-frame-pointer -
Werror=format-security -ansi -std=c99 -std=c11 -std=c18   -
D_POSIX_C_SOURCE=201112L -D_DEFAULT_SOURCE -D_GNU_SOURCE -fprofile-arcs -ftest-
coverage -lgcov test2.c -o test2_gcov
-------------- Running via our wrapper lcov_gen.sh --------------
lcov_gen.sh test2_gcov "1"
...
```



```
>>> lcov --capture --initial --directory . --output-file 0lcov_meta/lcovmeta_20240407_162405/appbase.info
Capturing coverage data from .
geninfo cmd: '/usr/local/bin/geninfo . --output-filename 0lcov_meta/lcovmeta_20240407_162405/appbase.info --ini
tial --memory 0 --branch-coverage'
Found gcov version: 13.2.0
Using intermediate gcov format
Writing temporary data to /tmp/geninfo_datiD6V
Scanning . for .gcno files ...
Found 1 graph files in .
Processing ./test2_gcov-test2.gcno
Finished .info-file creation
>>> eval ./test2_gcov 1
case 1
 In test2.c:foo():10
>>> lcov --capture --directory . --output-file 0lcov_meta/lcovmeta_20240407_162405/apptest.info
Capturing coverage data from .
geninfo cmd: '/usr/local/bin/geninfo . --output-filename 0lcov_meta/lcovmeta_20240407_162405/apptest.info --mem
ory 0 --branch-coverage'
Found gcov version: 13.2.0
Using intermediate gcov format
Writing temporary data to /tmp/geninfo_datZ3_3
Scanning . for .gcda files ...
Found 1 data files in .
Processing ./test2_gcov-test2.gcda
Finished .info-file creation
>>> lcov --add-tracefile 0lcov_meta/lcovmeta_20240407_162405/appbase.info --add-tracefile 0lcov_meta/lcovmeta_2
0240407_162405/apptest.info    --output-file 0lcov_meta/lcovmeta_20240407_162405/appfinal.info
Combining tracefiles.
.. found 2 files to aggregate.
Merging 0lcov_meta/lcovmeta_20240407_162405/appbase.info..1 remaining
Merging 0lcov_meta/lcovmeta_20240407_162405/apptest.info..0 remaining
Writing data to 0lcov_meta/lcovmeta_20240407_162405/appfinal.info
Summary coverage rate:
  lines......: 36.7% (11 of 30 lines)
  functions..: 50.0% (2 of 4 functions)
  branches...: 33.3% (2 of 6 branches)
>>> genhtml -o lcov_onerun_html/ -f -t Lcov: /big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_d
ebug/coverage_test 0lcov_meta/lcovmeta_20240407_162405/appfinal.info
Found 1 entries.
Found common filename prefix "/big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug"
Generating output.
Processing file coverage_test/test2.c
  lines=30 hit=11 functions=4 hit=2 branches=6 hit=2
Overall coverage rate:
  lines......: 36.7% (11 of 30 lines)
  functions......: 50.0% (2 of 4 functions)
  branches......: 33.3% (2 of 6 branches)
```

...

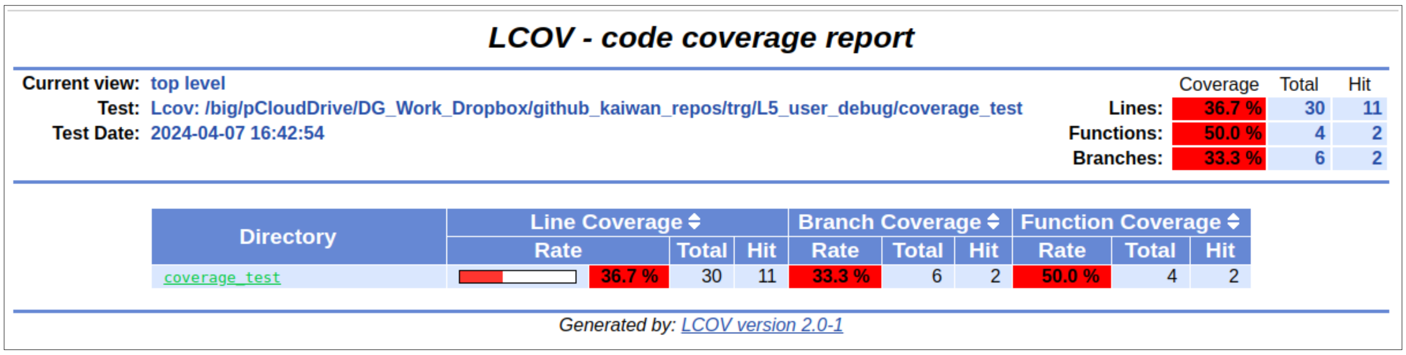


```
>>> genhtml -o lcov_merged_html/ -f -t "Lcov: /big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_
debug/coverage_test" merged.info
Found 1 entries.
Found common filename prefix "/big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug"
Generating output.
Processing file coverage_test/test2.c
  lines=30 hit=11 functions=4 hit=2 branches=6 hit=2
Overall coverage rate:
  lines......: 36.7% (11 of 30 lines)
  functions......: 50.0% (2 of 4 functions)
  branches......: 33.3% (2 of 6 branches)

Done.
-------------------------------- NOTE -------------------------------------
- If you want a cumulative / merged code coverage report, run your next coverage
test case via this script. In effect, simply adjust the CMDLINE_ARGS variable in
the 'better' Makefile and run 'make covg' again

- If you want to start from scratch, *wiping* previous coverage data, then
run this script with the -r (reset) option (you can add this option in the
Makefile invoking it if you wish to).
---------------------------------------------------------------------------
Once all coverage test cases are run, see the final report here:
 firefox file:///big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug/coverage_test/lcov_merge
d_html/index.html
 or
 google-chrome file:///big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug/coverage_test/lcov
_merged_html/index.html
coverage_test $
```

```
coverage_test $ ls
0lcov_meta/        lcov_onerun_html/   merged.info   test2_gcov*          test2_gcov-test2.gcno
lcov_merged_html/  Makefile            test2.c       test2_gcov-test2.gcda  test.c
coverage_test $
coverage_test $ l 0lcov_meta/
total 4.0K
drwxrwxr-x 2 kaiwan kaiwan 4.0K Apr  7 16:24 lcovmeta_20240407_162405/
coverage_test $
```

The lcov meta dirs can be seen

Lookup first run code coverage results via a web browser:



LCOV - code coverage report

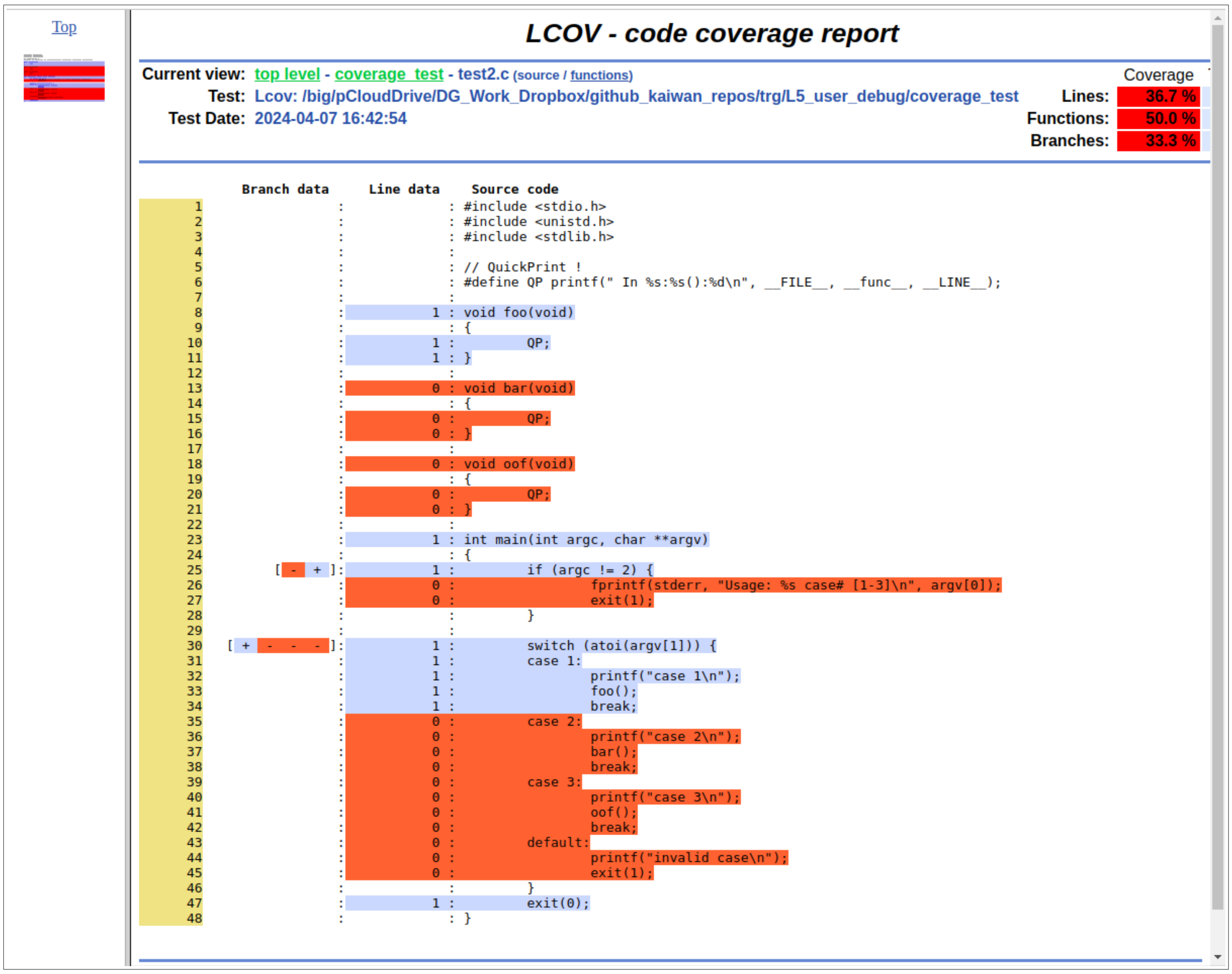
Current view: top level
Test: Lcov: /big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug/coverage_test
Test Date: 2024-04-07 16:42:54

	Coverage	Total	Hit
Lines:	36.7 %	30	11
Functions:	50.0 %	4	2
Branches:	33.3 %	6	2

Directory	Line Coverage			Branch Coverage			Function Coverage		
	Rate	Total	Hit	Rate	Total	Hit	Rate	Total	Hit
coverage_test	36.7 %	30	11	33.3 %	6	2	50.0 %	4	2

Generated by: LCOV version 2.0-1

Detailed view – click on the *coverage_test* and then on the *test2.c* hyperlinks:



LCOV - code coverage report

Current view:	top level - coverage_test - test2.c (source / functions)		Coverage
Test:	Lcov: /big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug/coverage_test	Lines:	36.7 %
Test Date:	2024-04-07 16:42:54	Functions:	50.0 %
		Branches:	33.3 %

```
     Branch data     Line data    Source code
                  :              : #include <stdio.h>
                  :              : #include <unistd.h>
                  :              : #include <stdlib.h>
                  :              :
                  :              : // QuickPrint !
                  :              : #define QP printf(" In %s:%s():%d\n", __FILE__, __func__, __LINE__);
                  :              :
                  :            1 : void foo(void)
                  :              : {
                  :            1 :         QP;
                  :            1 : }
                  :              :
                  :            0 : void bar(void)
                  :              : {
                  :            0 :         QP;
                  :            0 : }
                  :              :
                  :            0 : void oof(void)
                  :              : {
                  :            0 :         QP;
                  :            0 : }
                  :              :
                  :            1 : int main(int argc, char **argv)
                  :              : {
           [ - + ]:            1 :         if (argc != 2) {
                  :            0 :                 fprintf(stderr, "Usage: %s case# [1-3]\n", argv[0]);
                  :            0 :                 exit(1);
                  :              :         }
                  :              :
   [ + - - - ]:            1 :         switch (atoi(argv[1])) {
                  :            1 :         case 1:
                  :            1 :                 printf("case 1\n");
                  :            1 :                 foo();
                  :            1 :                 break;
                  :            0 :         case 2:
                  :            0 :                 printf("case 2\n");
                  :            0 :                 bar();
                  :            0 :                 break;
                  :            0 :         case 3:
                  :            0 :                 printf("case 3\n");
                  :            0 :                 oof();
                  :            0 :                 break;
                  :            0 :         default:
                  :            0 :                 printf("invalid case\n");
                  :            0 :                 exit(1);
                  :              :         }
                  :            1 :         exit(0);
                  :              : }
```

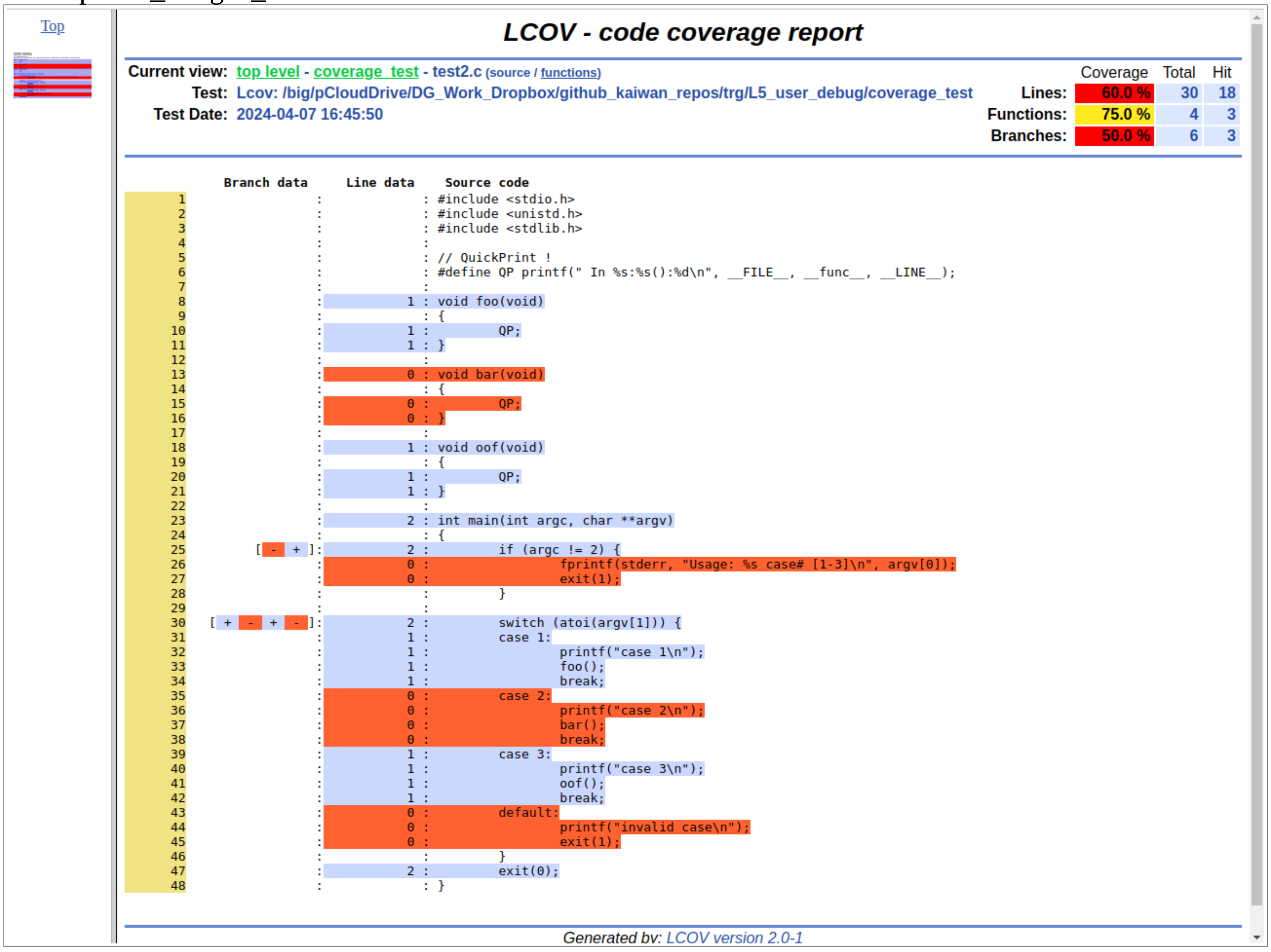
Fantastic! Can literally see the lines covered - and the number of times they've run (left col, blue background), the lines that haven't been covered (red background), as well as the branches (not) taken. (This time, we ran it with parameter 1, hence only the case 1 code got hit.)

Run #2 of 3: with CMDLINE_ARGS="3"

```
$ cat Makefile
...
FNAME_C := test2
#--- CHECK: manually add params as required
# Populate any required cmdline arguments to the process here:
CMDLINE_ARGS="3"
...

$ make covg
...
...
Processing file coverage_test/test2.c
  lines=30 hit=18 functions=4 hit=3 branches=6 hit=3
Overall coverage rate:
  lines......: 60.0% (18 of 30 lines)
  functions......: 75.0% (3 of 4 functions)
  branches......: 50.0% (3 of 6 branches)

Done.
...
```

Aha! More coverage!

Lookup lcov_merged_html/index.html now:



Top

LCOV - code coverage report

Current view: top level - coverage_test - test2.c (source / functions)
Test: Lcov: /big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug/coverage_test
Test Date: 2024-04-07 16:45:50

	Coverage	Total	Hit
Lines:	60.0 %	30	18
Functions:	75.0 %	4	3
Branches:	50.0 %	6	3

```
         Branch data     Line data    Source code
                   :              : #include <stdio.h>
                   :              : #include <unistd.h>
                   :              : #include <stdlib.h>
                   :              :
                   :              : // QuickPrint !
                   :              : #define QP printf(" In %s:%s():%d\n", __FILE__, __func__, __LINE__);
                   :              :
                   :            1 : void foo(void)
                   :              : {
                   :            1 :         QP;
                   :            1 : }
                   :              :
                   :            0 : void bar(void)
                   :              : {
                   :            0 :         QP;
                   :            0 : }
                   :              :
                   :            1 : void oof(void)
                   :              : {
                   :            1 :         QP;
                   :            1 : }
                   :              :
                   :            2 : int main(int argc, char **argv)
                   :              : {
             [ - + ]:            2 :         if (argc != 2) {
                   :            0 :                 fprintf(stderr, "Usage: %s case# [1-3]\n", argv[0]);
                   :            0 :                 exit(1);
                   :              :         }
                   :              :
       [ + - + - ]:            2 :         switch (atoi(argv[1])) {
                   :            1 :         case 1:
                   :            1 :                 printf("case 1\n");
                   :            1 :                 foo();
                   :            1 :                 break;
                   :            0 :         case 2:
                   :            0 :                 printf("case 2\n");
                   :            0 :                 bar();
                   :            0 :                 break;
                   :            1 :         case 3:
                   :            1 :                 printf("case 3\n");
                   :            1 :                 oof();
                   :            1 :                 break;
                   :            0 :         default:
                   :            0 :                 printf("invalid case\n");
                   :            0 :                 exit(1);
                   :              :         }
                   :            2 :         exit(0);
                   :              : }
```

Generated by: LCOV version 2.0-1

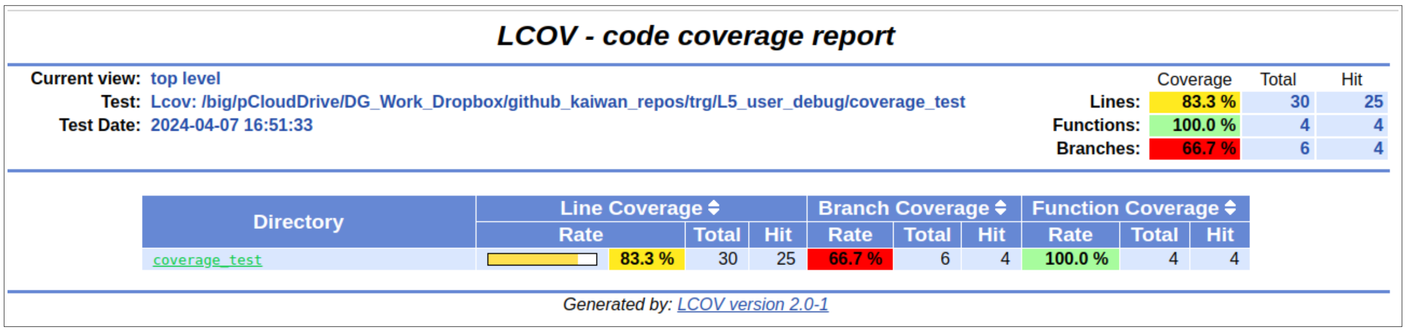
Run #3 of 3: with CMDLINE_ARGS="2"

```
$ cat Makefile
...
FNAME_C := test2
#--- CHECK: manually add params as required
# Populate any required cmdline arguments to the process here:
CMDLINE_ARGS="2"
...

$ make covg
...
...
Processing file coverage_test/test2.c
```

```
  lines=30 hit=25 functions=4 hit=4 branches=6 hit=4
Overall coverage rate:
  lines......: 83.3% (25 of 30 lines)
  functions......: 100.0% (4 of 4 functions)
  branches......: 66.7% (4 of 6 branches)

Done.
...
```

More coverage!

Lookup lcov_merged_html/index.html now:



LCOV - code coverage report

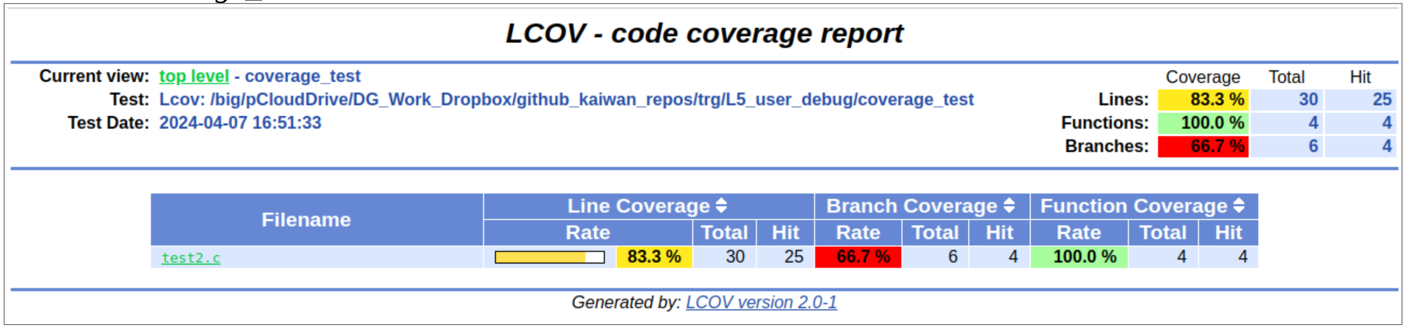
Current view: top level
Test: Lcov: /big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug/coverage_test
Test Date: 2024-04-07 16:51:33

	Coverage	Total	Hit
Lines:	83.3 %	30	25
Functions:	100.0 %	4	4
Branches:	66.7 %	6	4

Directory	Line Coverage				Branch Coverage			Function Coverage		
	Rate		Total	Hit	Rate	Total	Hit	Rate	Total	Hit
coverage_test		83.3 %	30	25	66.7 %	6	4	100.0 %	4	4

Generated by: *LCOV version 2.0-1*

Click the *coverage_test* link:



LCOV - code coverage report

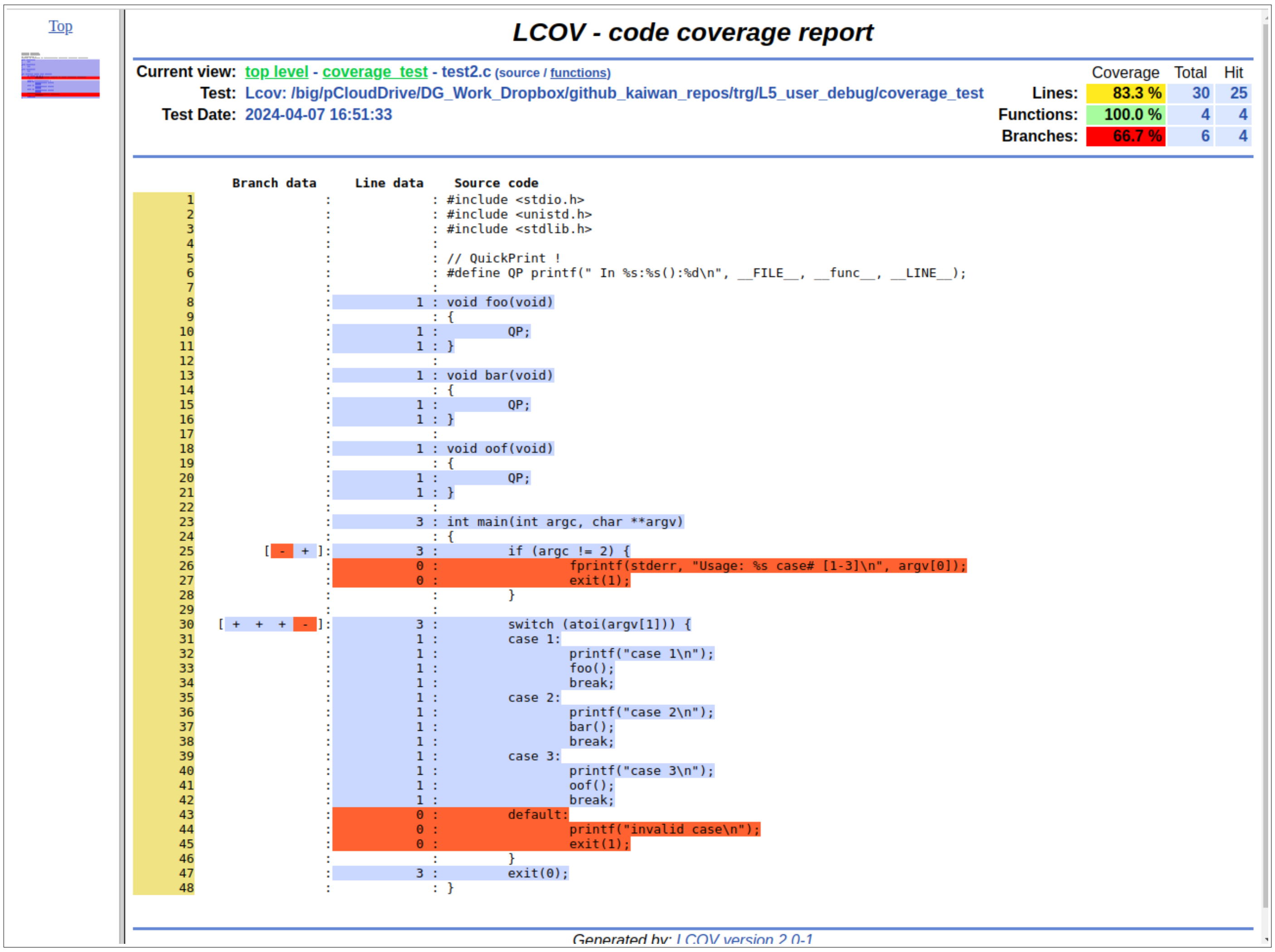
Current view: top level - coverage_test
Test: Lcov: /big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug/coverage_test
Test Date: 2024-04-07 16:51:33

	Coverage	Total	Hit
Lines:	83.3 %	30	25
Functions:	100.0 %	4	4
Branches:	66.7 %	6	4

Filename	Line Coverage				Branch Coverage			Function Coverage		
	Rate		Total	Hit	Rate	Total	Hit	Rate	Total	Hit
test2.c		83.3 %	30	25	66.7 %	6	4	100.0 %	4	4

Generated by: *LCOV version 2.0-1*

Click the *test2.c* link:


Top

LCOV - code coverage report

Current view: top level - coverage_test - test2.c (source / functions)
Test: Lcov: /big/pCloudDrive/DG_Work_Dropbox/github_kaiwan_repos/trg/L5_user_debug/coverage_test
Test Date: 2024-04-07 16:51:33

	Coverage	Total	Hit
Lines:	83.3 %	30	25
Functions:	100.0 %	4	4
Branches:	66.7 %	6	4

```
Branch data     Line data    Source code
                  :             : #include <stdio.h>
                  :             : #include <unistd.h>
                  :             : #include <stdlib.h>
                  :             :
                  :             : // QuickPrint !
                  :             : #define QP printf(" In %s:%s():%d\n", __FILE__, __func__, __LINE__);
                  :             :
                  :           1 : void foo(void)
                  :             : {
                  :           1 :         QP;
                  :           1 : }
                  :             :
                  :           1 : void bar(void)
                  :             : {
                  :           1 :         QP;
                  :           1 : }
                  :             :
                  :           1 : void oof(void)
                  :             : {
                  :           1 :         QP;
                  :           1 : }
                  :             :
                  :           3 : int main(int argc, char **argv)
                  :             : {
        [ - + ]:           3 :         if (argc != 2) {
                  :           0 :                 fprintf(stderr, "Usage: %s case# [1-3]\n", argv[0]);
                  :           0 :                 exit(1);
                  :             :         }
                  :             :
  [ + + + - ]:           3 :         switch (atoi(argv[1])) {
                  :           1 :         case 1:
                  :           1 :                 printf("case 1\n");
                  :           1 :                 foo();
                  :           1 :                 break;
                  :           1 :         case 2:
                  :           1 :                 printf("case 2\n");
                  :           1 :                 bar();
                  :           1 :                 break;
                  :           1 :         case 3:
                  :           1 :                 printf("case 3\n");
                  :           1 :                 oof();
                  :           1 :                 break;
                  :           0 :         default:
                  :           0 :                 printf("invalid case\n");
                  :           0 :                 exit(1);
                  :             :         }
                  :           3 :         exit(0);
                  :             : }
```

Generated by: LCOV version 2.0-1

All right! We can now literally see that most of the code lines (83.3%) have been covered, and, more importantly, which lines remains to be covered.

It's critical to now write appropriate test cases / use fault injection / whatever .. to actually get all lines covered, i.e., to meet the goal of 100% code coverage!

And then ... even more important - to run all your (dynamic analysis) tests via these test cases so that all lines of code get exercised !

GOAL: execute every line of code!

Write test cases to ensure this. Run this entire suite of test cases with both static and dynamic analysis tooling turned on.

Merging of several gcov test runs to get a comprehensive coverage report is supported by lcov!

The key is using lcov to generate an aggregated coverage report by invoking it like this:

```
lcov  \
  --add-tracefile </path/to/xxxfinal1.info> --test-name whatever1 \
  --add-tracefile </path/to/xxxfinal2.info> --test-name whatever2 \
  ... \
  --output-file merged.info
```

The .info files are known as coverage tracefiles.
Our *lcov_gen.sh* wrapper script does this automatically! In our previous test2 example – the *lcov_gen.sh* script does this:

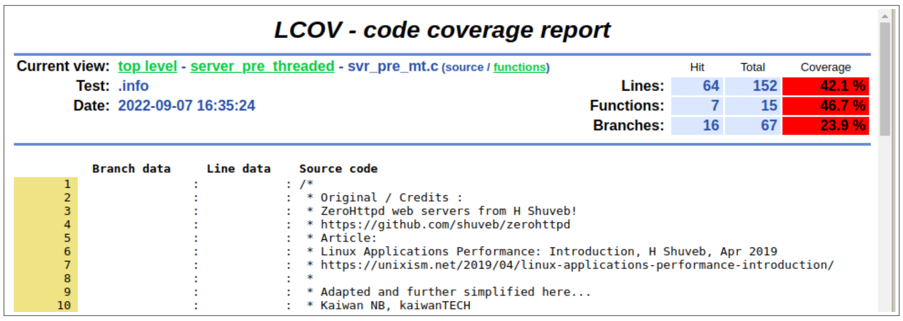
```
lcov  \
  --add-tracefile 0lcov_meta/lcovmeta_20240407_164252/appfinal.info --test-name
lcovmeta_20240407_164252 \
  --add-tracefile 0lcov_meta/lcovmeta_20240407_164549/appfinal.info --test-name
lcovmeta_20240407_164549 \
  --add-tracefile 0lcov_meta/lcovmeta_20240407_165132/appfinal.info --test-name
lcovmeta_20240407_165132 \
  --output-file merged.info
```

To disable / start with a clean slate, run it with the -r (reset) option switch:

```
$ lcov_gen.sh -h
Usage: lcov_gen.sh [-r] app-under-test-pathname arg1 arg2 [...]
 -r : RESET mode: when you pass -r, all existing lcov metadata is deleted,
      in effect giving you a fresh start. DON'T pass it if you'intending to
      run several code coverage test cases one by one, in order to generate a
      merged code coverage report.
```

Ref: https://backstreetcoder.com/code-coverage-using-gcov-lcov-in-linux/

As one more example of using gcov/lcov (run on the network server *net_sockets/inet_domain/server_pre_threaded*):

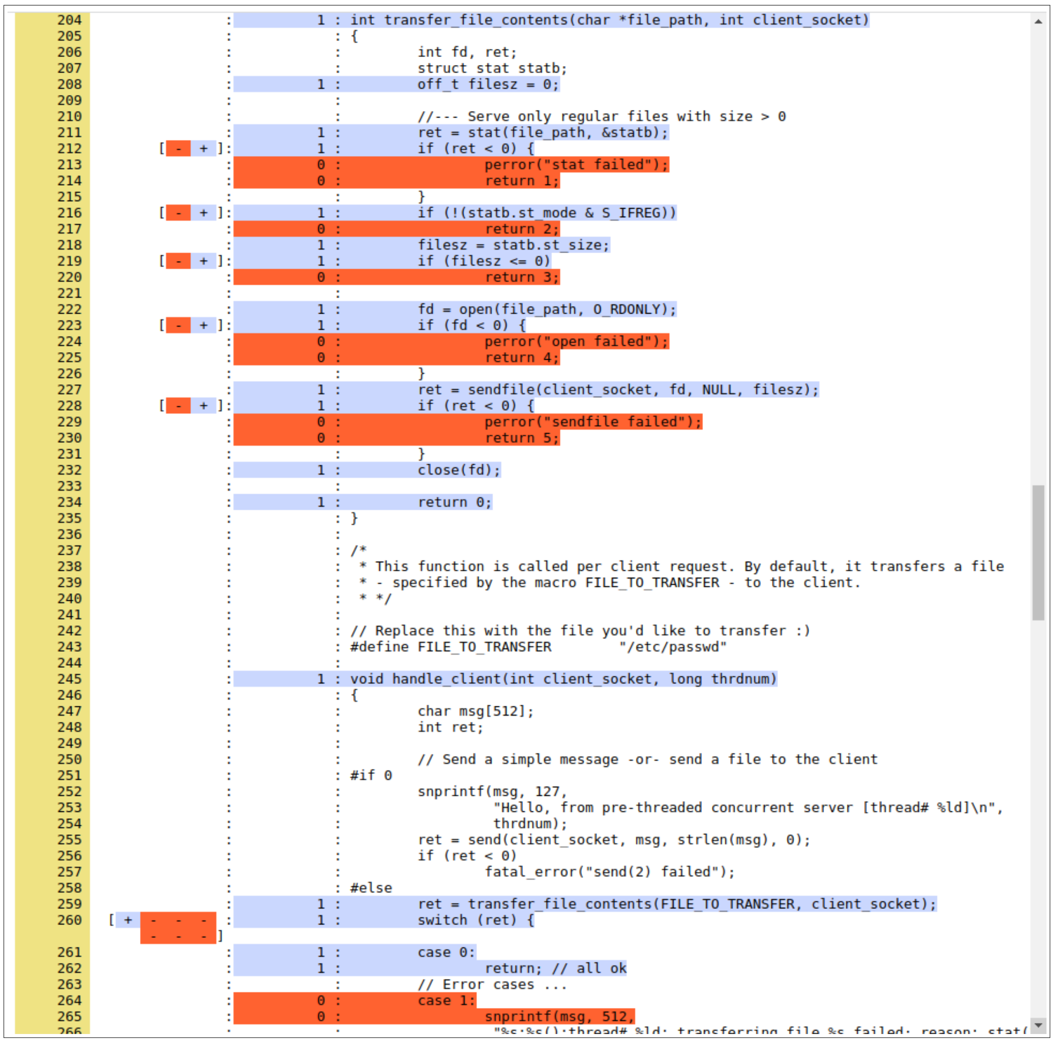


LCOV - code coverage report

Current view: top level - server_pre_threaded - svr_pre_mt.c (source / functions)
Test: .info
Date: 2022-09-07 16:35:24

	Hit	Total	Coverage
Lines:	64	152	42.1 %
Functions:	7	15	46.7 %
Branches:	16	67	23.9 %

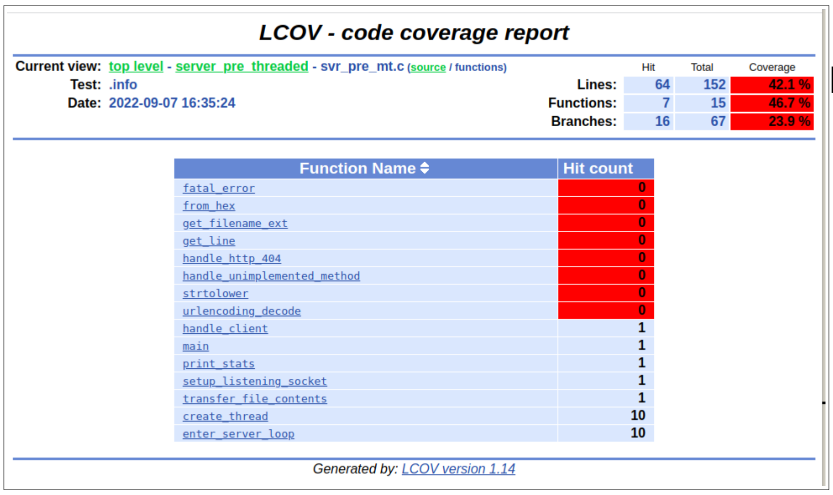
```
   Branch data     Line data    Source code

               :              : /*
               :              :  * Original / Credits :
               :              :  * ZeroHttpd web servers from H Shuveb!
               :              :  * https://github.com/shuveb/zerohttpd
               :              :  * Article:
               :              :  * Linux Applications Performance: Introduction, H Shuveb, Apr 2019
               :              :  * https://unixism.net/2019/04/linux-applications-performance-introduction/
               :              :  *
               :              :  * Adapted and further simplified here...
               :              :  * Kaiwan NB, kaiwanTECH
```



```
               :            1 : int transfer_file_contents(char *file_path, int client_socket)
               :              : {
               :              :         int fd, ret;
               :              :         struct stat statb;
               :            1 :         off_t filesz = 0;
               :              :
               :              :         //--- Serve only regular files with size > 0
               :            1 :         ret = stat(file_path, &statb);
      [ - + ]:            1 :         if (ret < 0) {
               :            0 :                 perror("stat failed");
               :            0 :                 return 1;
               :              :         }
      [ - + ]:            1 :         if (!(statb.st_mode & S_IFREG))
               :            0 :                 return 2;
               :            1 :         filesz = statb.st_size;
      [ - + ]:            1 :         if (filesz <= 0)
               :            0 :                 return 3;
               :              :
               :            1 :         fd = open(file_path, O_RDONLY);
      [ - + ]:            1 :         if (fd < 0) {
               :            0 :                 perror("open failed");
               :            0 :                 return 4;
               :              :         }
               :            1 :         ret = sendfile(client_socket, fd, NULL, filesz);
      [ - + ]:            1 :         if (ret < 0) {
               :            0 :                 perror("sendfile failed");
               :            0 :                 return 5;
               :              :         }
               :            1 :         close(fd);
               :              :
               :            1 :         return 0;
               :              : }
               :              :
               :              : /*
               :              :  * This function is called per client request. By default, it transfers a file
               :              :  * - specified by the macro FILE_TO_TRANSFER - to the client.
               :              :  * */
               :              :
               :              : // Replace this with the file you'd like to transfer :)
               :              : #define FILE_TO_TRANSFER        "/etc/passwd"
               :              :
               :            1 : void handle_client(int client_socket, long thrdnum)
               :              : {
               :              :         char msg[512];
               :              :         int ret;
               :              :
               :              :         // Send a simple message -or- send a file to the client
               :              : #if 0
               :              :         snprintf(msg, 127,
               :              :                 "Hello, from pre-threaded concurrent server [thread# %ld]\n",
               :              :                 thrdnum);
               :              :         ret = send(client_socket, msg, strlen(msg), 0);
               :              :         if (ret < 0)
               :              :                 fatal_error("send(2) failed");
               :              : #else
               :            1 :         ret = transfer_file_contents(FILE_TO_TRANSFER, client_socket);
  [ + - - -   :            1 :         switch (ret) {
     - - - ]
               :            1 :         case 0:
               :            1 :                 return; // all ok
               :              :         // Error cases ...
               :            0 :         case 1:
               :            0 :                 snprintf(msg, 512,
               :              :                 "%s:%s():thread# %ld: transferring file %s failed: reason: stat(
```

[…]

Clicking on the ‘functions’ link:



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - server_pre_threaded - svr_pre_mt.c (source / functions)				
Test:	.info	Lines:	64	152	42.1 %
Date:	2022-09-07 16:35:24	Functions:	7	15	46.7 %
		Branches:	16	67	23.9 %

Function Name ⇕	Hit count
fatal_error	0
from_hex	0
get_filename_ext	0
get_line	0
handle_http_404	0
handle_unimplemented_method	0
strtolower	0
urlencoding_decode	0
handle_client	1
main	1
print_stats	1
setup_listening_socket	1
transfer_file_contents	1
create_thread	10
enter_server_loop	10

Generated by: LCOV version 1.14

[…]

[End document]