Dumping gcov data at runtime - a simple example

Der Herr Hofrat

OpenTech EDV Research GmbH, Austria
< der.herr@hofr.at >

Revision History	
Revision 0.1	2011-07-27
First release	

Table of Contents

1. Introduction

The gnu code coverage tool **gcov** is generally used to dump the coverage data when the program exits - for some embedded systems or for server processes you need a means to dump coverage data of particular situations, i.e. when a client connects. To achieve this, one might need to dump the coverage data triggered by a runtime condition in the code or by an external signal. In this HOWTO, we briefly describe how to kick **gcov** at runtime. It seems that this is not quite the intended use and maybe the **gcov** maintainer would consider making **gcov_flush** available in a more direct way for such use cases. Nevertheless, it is not really that wild a business to get access to the data in a clean way - which we demonstrate here based on a trivial infinite loop example. What this is good for is to get the coverage data of a specific portion of the code at execution time, for debugging purposes, for analysis (test coverage) or to optimize for a particular case by feeding the relevant part back to the compiler with **-fbranch-probabilities**.

This example was running on a stock Debian squeeze 6.0.2 with the default toolchain.

2. Preparatory work

2.1. Check for a possible version mismatch

In a first step, make sure that you will be using the proper **libgcov** that fits your compiler - check with **gcc --version** if unsure.

```
hofrat@debian:~/gcov$ gcc --version
gcc (Debian 4.3.2-1.1) 4.3.2
Copyright (C) 2008 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

hofrat@debian:~/gcov$ ls /usr/lib/gcc/x86_64-linux-gnu/4.3/libgcov.a
/usr/lib/gcc/x86_64-linux-gnu/4.3/libgcov.a
```

2.2. Get rid of annoying warnings

Note that even if you link with **-lgcov**, **gcc** will fuss at you about:

```
hello.c: In function ’my_handler’:
hello.c:17: warning: implicit declaration of function ’__gcov_flush’
```

To get around this, check the prototype of **__gcov_flush** in the **gcc** sources in **gcc/gcov-io.h** and add a proper declaration to your source file.

3. Example hello.c

3.1. Compiling, linking and running the example application

The **-fprofile-arcs** is the flag to instrument the code, basically adding a 64-bit counter in each basic-block, **-ftest-coverage** tells **gcc** to dump the notes file for each source file as **SOURCE.gcno**. For details check **man 1 gcov**.

```
hofrat@debian:~/gcov$ gcc -Wall -fprofile-arcs -ftest-coverage hello.c -o hello

hofrat@debian:~/gcov$ ls
hello  hello.c  hello.gcno
```

The process is started as background process so that we can examplify how to inspect it using our signal handler and **gcov** at runtime.

```
hofrat@debian:~/gcov$ ./hello &
[1] 5164
```

3.2. Dumping and analyzing the coverage data

Sending the **SIGUSR1** signal to the **hello** process dumps the **gcov** data, and we can generate the coverage with **gcov** using the intermediate data. The runtime data are dumped into **hello.gcda** (**hello.gcno** was generated during compilation - for details see the **gcov** man page).

```
hofrat@debian:~/gcov$ killall -USR1 hello
received signal
2514147346
```

Use **gcov** for analysis:

```
hofrat@debian:~/gcov$ gcov hello
File 'hello.c'
Lines executed:100.00% of 14
hello.c:creating 'hello.c.gcov'

hofrat@debian:~/gcov$ tail hello.c.gcov
        1:   34:  new_action.sa_flags = 0;
        -:   35:
        1:   36:  sigaction(SIGUSR1, NULL, &old_action);
        1:   37:  if (old_action.sa_handler != SIG_IGN)
        1:   38:    sigaction (SIGUSR1, &new_action, NULL);
        -:   39:
        -:   40:  /* infinite loop - to exemplify dumping coverage data while program runs */
2514147346:   41:  for(n = 0; ; n++)
5028294692:   42:    i++;
        -:   43:}
```

As only the runtime loop is of interest in this trivial example we simply use **tail** on the generated hello.c.gcov file to show that the runtime data is updated. Note that there is of course no strict synchronization, and the runtime data in this case roughly reflect the state of affairs at the time the signal was received.

The next sequence is just a repetition showing that the data at runtime can be updated effectively without terminating the process. Note though that reading the counters in this state is in no way synchronized, so be careful with interpretation of the data.

```
hofrat@debian:~/gcov$ killall -USR1 hello
received signal
186275468151

hofrat@debian:~/gcov$ gcov hello
File 'hello.c'
Lines executed:100.00% of 15
hello.c:creating 'hello.c.gcov'
```

```
hofrat@debian:~/gcov$ tail hello.c.gcov
        1:   34:  new_action.sa_flags = 0;
        -:   35:
        1:   36:  sigaction(SIGUSR1, NULL, &old_action);
        1:   37:  if (old_action.sa_handler != SIG_IGN)
        1:   38:    sigaction (SIGUSR1, &new_action, NULL);
        -:   39:
        -:   40:  /* infinite loop - to exemplify dumping coverage data while program runs */
186275468151:   41:  for(n = 0; ; n++)
372550936302:   42:    i++;
        -:   43:}
```

3.3. Conclusion

So now you can query the **gcov** data at runtime with a simple **SIGUSR1** signal and then convert it with **gcov** or **lcov** as you like. No need to invent anything new. Code example below. This might not be the most elegant solution but I guess it should solve the problem of not wanting to wait until the end of the program to inspect coverage as well as getting specific coverage data for a code section of interest. You of course could also dump the coverage data from your program by simply calling **__gcov_flush();** if some condition occurs.

3.4. Example code

The wild and daring code example is as found below - note that the **__gcov_flush()** function is not documented in a man page or the like - if you want to know details, download the **gcc** sources and check the files in **gcc/gcov_*.c** for details.

The symbol **__gcov_flush** is available as a global symbol in **libgcov.a** as can be seen by inspection with **nm**:

```
hofrat@debian:~/gcov$ nm --print-armap libgcov.a | grep gcov_flush
__gcov_flush in _gcov.o
00000000000013c0 T __gcov_flush
...
```

This might not be the most elegant way to do it - but it seems like this problem is not really described anywhere - at least I did not find a description of this solution in a reasonable time. It also may well be **gcc** version dependent so you might need to adjust the actual routine to be called in the signal handler depending on the **gcc** version in use.

4. Listing

```
/* dumping gcov data at runtime - note: no error handling to keep it simple */
/* Compile: gcc -Wall -fprofile-arcs -ftest-coverage hello.c -o hello
 * Run: ./hello &
 *      kill -s SIGUSR1 ‘pidof hello‘
 *
 * Author: Der Herr Hofr.at
 * Copyright: OpenTech EDV Research GmbH 2011
 * License: GPL V2
 */
#include <stdio.h>
```

```
#include <signal.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

static unsigned long long i = 0;
void __gcov_flush(void); /* check in gcc sources gcc/gcov-io.h for the prototype */

void my_handler(int signum)
{
  printf("received signal\n");
  printf("%llu\n", i);
  __gcov_flush(); /* dump coverage data on receiving SIGUSR1 */
}

int main(int argc, char **argv)
{
  struct sigaction new_action, old_action;
  int n;

  /* setup signal hander */
  new_action.sa_handler = my_handler;
  sigemptyset(&new_action.sa_mask);
  new_action.sa_flags = 0;

  sigaction(SIGUSR1, NULL, &old_action);
  if (old_action.sa_handler != SIG_IGN)
    sigaction (SIGUSR1, &new_action, NULL);

  /* infinite loop - to exemplify dumping coverage data while program runs */
  for(n = 0; ; n++)
    i++;
}
```