GCOV LEON USER & INSTALLATION GUIDE

To	:	**Applications and Boot Developpers on LEON3 Architecture**
From	:	Dupeyron Jérôme
Phone	:	0534363299
Fax	:	**N/A**
Date	:	13 March 2014
Subject	:	Installation and User Guide Gcov Library
Ref.	:	GcovForLEON3InstallationAndUserGuideV1A
C.C.	:	

1 Purpose of this document

This document's purpose is to provide an installation and usage guide for the tool and the library allowing the usage of gcov instrumentation on LEON3 real or simulated targets.

2 Installation guide

2.1 Prerequisites

The tool works with any compiler able to generate instrumented code compatible with gcov libraries derived from gcc versions 3.x or 4.x.
The tool needs python version above 2.5 to be able to work.

2.2 Installation procedure

The product is available in the form of a tarball file. Just unzip and untar it to a directory in your filesystem.

```
-bash-3.2$ tar -xvf 01A.tar
01A/
01A/run3/
01A/README
01A/COTS/
01A/COTS/gcovr-3.1/
<… Truncated, more lines for files included in gcovr-3.1 distribution…>
01A/COTS/README
01A/COTS/gcovr-3.1.tar
01A/run1/
01A/tool/
01A/tool/createfile.py
01A/tool/overload.c
01A/Makefile
01A/inc/
01A/inc/thread2.h
01A/inc/thread1.h
01A/lib/
01A/src/
01A/src/thread1.c
```

```
01A/src/thread2.c
01A/src/multithread.c
01A/run2/
```

The files in orange above are the ones mandatory to use the tool, the others are an example done with gcc 4.4.

The directory COTS contains the open source tool gcovr version 3.1 downloaded from
`https://github.com/gcovr/gcovr/archive/3.1.tar.gz`

The Makefile in the main directory gives an example of use of the tool with gcc 4.4.

The directories src, inc, lib, run1, run2 and run3 are used to run the test of deployment by calling make all.

The normal execution is recorded in the README file.

If you get the same execution trace, the installation is successful.

3 Usage Guide

3.1 Gcov instrumentation

The principle of gcov instrumentation in version 3.x and 4.x of gcc is the following:

1. In the generated binary:

It adds a data structure of fixed size depending on the code complexity on each object file to hold the counters of branch activation.
It adds in the bootstrap of the application a call to initialize each data structure upon loading of the object file.
It adds a call to gcov_exit to exit() bootstrapped service.
It adds on each branch of code a call to increment the counter.

2. In the file system:

It creates a sourcefile.gcno for each source file which holds a binary static structure describing the structure of the code upon compilation with gcov options.
It creates a sourcefile.gcda for each source file which holds the counters data when the application instrumented is exited or forked. This file is located next to source file and is updated between multiple runs of the application.

3.2 LEON3 Embedded context

In embedded context, the following differences from the case above are addressed by this tool:

- The file system is not available
    - It prevents the recording of the values by the gcov library included in the instrumented binary
    - It prevents from updating data between multiple tests.
- The bootstraps are not standard application ones
    - The initialization of data might not be done upon application startup
    - The dump of data is not done as the application is not supposed to terminate.

3.3 Solution proposed by this tool

This tool is composed of two parts:

1. A library that will allow the record of the data at the end of the test and a tool converting the recorded data to gcov gcda file format.
2. A tool that will merge the coverage results between multiple gcda files.

3.3.1 Tool and library to record data of one test

3.3.1.1 Principle of work

The library used by gcov to record the data uses fopen, fwrite, fread, fclose services, the library composed of the object file overload.o provided in the tool overload those services so that the data is sent to standard output and can be postprocessed by the tool to regenerate the correct file in the execution directory.

3.3.1.2 The library

This library redefines the following services open, fopen, setbuf, fdopen, close, fclose, fread, fwrite, fseek and fcntl.
The redefinition allows the library to hold the name of the files opened and to generate data dump upon write commands.
It also defines services named min, cs, gendatastring and glboals fdall and arrayFILES holding the list of opened file descriptors.
Compile this file without gcov instrumentation and put the object resulting file in the link command line without including it in a library so that it's taken before the libc services.

When compiling the source file, you need to define two globals using the -D gcc option (see Makefile for example):

```
-DNBSOURCEFILES=100 : This option will allocate 100 FILE objects in memory, in order to limit the memory consumption, set this number to your number of source files instrumented by gcov.
-DSTRINGSIZE=256 : This option will define the maximum size of a string for the dump of data, adjust it to more or less depending on the data loss or corruption during dump, less will be more reliable but slower, more will be faster but error prone.
```

3.3.1.3 Instrumenting the source

```
When instrumenting the source files you need to pass the following options to your gcc line :
-ftest-coverage -fprofile-arcs

Note : overload.c file must not be instrumented.
```

3.3.1.4 Making the system work

3.3.1.4.1 Initializing the gcov library (optional)

If you use a specific bootstrap for starting your binary, it's necessary to create another bootstrap for instrumentation calling the initialisation service of the gcov instrumentation. This is done automatically

and silently by gcc when generating the binary with the instrumentation options but when the bootstrap is specific, it needs to be done manually.
The service to call is TBD.

3.3.1.4.2 Terminating and dumping

As it is embedded software, the exit service or fork service is propably not used by your code, so the instrumentation termination service will not be called automatically.
You need to define a clean termination service in your software that will not be in an instrumented code and will call the termination service of the library gcov, this service is either TBD or TBD.

3.3.1.4.3 Output of the library example

In the README, you have an example of dump of the data, if you don't get that, you may need to modify the overload.c file to activate the DEBUG #defined value to see if you have effective calls to fopen, fread etc services.

```
###NEWFD 0000 /home/jdupeyro/gcovtest/lib/multithread.gcda 4588###
###DATA 0000
6164636770313034d703605000000001020000000300000098e453470000a10122000000010000000000
0000000000000000000000010000000000000000000000000000000100000000000000000000
0000000000000010000000000000010000000000000000000000000000000100000000000000000000
0000000000 2421###
###DATA 0000
0000000000000000000000000000000000000000000000000000000000000000000000000000000010
000000000000000000a109000000000000011000000100000007000000000000000010
0000000000000010000000000000000000a309000000fca57c4f190000000100000034010000000000
0064000000 2565###
###DATA 0000 00000000640000000000000000000000 1770###
###CLOSE 0000 0192###
###NEWFD 0001 /home/jdupeyro/gcovtest/lib/thread1.gcda 4083###
###DATA 0001
6164636770313034 1e046050000000102000000030000 0d94aae2f0000a10108000000000000000000
0000000000000000000000000000000000000000000000000000000a109000000000
0000004000000010000000000000000000000000000000000000000000000000000000000a3090000
00fca57c4f 2995###
###DATA 0001
190000000100000034010000000000006400000000000000640000000000000000000000 3720###
###CLOSE 0001 0193###
###NEWFD 0002 /home/jdupeyro/gcovtest/lib/thread2.gcda 4085###
###DATA 0002
6164636770313034 5a046050000000102000000030000 0014e835d80000a10108000000064000000000
0000064000000000000000640000000000000001000000000000000000000a109000000000
0000004000000010000002d010000000000006400000000000000640000000000000000000000a3090000
00fca57c4f 2964###
###DATA 0002
190000000100000034010000000000006400000000000000640000000000000000000000 3721###
###CLOSE 0002 0194###
```

The data is dumped with ascii (code between 32 and 127) characters and is composed of three types of traces :
NEWFD which will associate a file descriptor (0000) with a file name, the last value is a checksum allowing data corruption detection.
DATA which will associate a file descriptor (0000) with a packet of data, a checksum is also provided.
CLOSE which indicate the correct end of dump for the file descriptor provided.

3.3.1.5 Tool usage

The tool will use the standard output capture file to generate the gcda files.

It's usage is very simple: `cat trace | <DeploymentDirectory>/tool/createfile.py` where trace has been a capture of the standard output done by your preferred tool

The tool must be run on the filesystem and machine where you made the instrumentation gcov.
The capture of the data can be done on any machine connected to the embedded environment for trace capture.

This will generate in the current directory the gcda files dumped. One for each source file instrumented.

After processing by the tool, all the lines dumped by the tool should be displayed with a leading CORRECT word indicating the postprocessing tool identified a correct data.
If the format was too corrupted for the tool, nothing will be displayed, if data was corrupted but understood by the tool, incorrect lines will be displayed with a leading INCORRECT word.

The tool also handle the state of file descriptors and checks that a fd cannot be opened twice at a time and should have been opened before closing.
In those cases the following lines should appear in the trace :
`INVALID UNKNOWN FD + line : the file descriptor on which we try to write or to close is not opened at that time.`
`INVALID ALREADY OPEN FD + line : the file description that we try to open is already opened.`

3.3.2 Tool to merge results

3.3.2.1 Open source and license

The tool used is an open source tool named gcovr version 3.1.
The licence is under a BSD licence.

3.3.2.2 Usage guide

The complete usage guide of the tool can be found at http://gcovr.com/guide.html for the latest version if you want to go deeper in the tool.
The rest of the paragraph details the basic usage already tested and only supported for the perimeter of this embedded LEON3 environment software.

3.3.2.3 Setting up the environment

In order to be able to use the synthesis tool, it's necessary to setup a virtual environment with the following structure (You have an example in the run1, run2 and run3 directories) one for each of your tests execution:

The tool is assuming that you have unique source file names throughout all your code even in different directories.

In the run1 directory I have the following files before running gcovr :

```
-bash-3.2$ ll -R .
.:
total 12
-rw-r--r-- 1 jdupeyro users 264 2014-03-13 10:19 multithread.gcda
lrwxrwxrwx 1 jdupeyro users  23 2014-03-13 10:19 multithread.gcno ->
../lib/multithread.gcno
lrwxrwxrwx 1 jdupeyro users   6 2014-03-13 10:19 src -> ../src
-rw-r--r-- 1 jdupeyro users 160 2014-03-13 10:19 thread1.gcda
lrwxrwxrwx 1 jdupeyro users  19 2014-03-13 10:19 thread1.gcno ->
../lib/thread1.gcno
-rw-r--r-- 1 jdupeyro users 160 2014-03-13 10:19 thread2.gcda
lrwxrwxrwx 1 jdupeyro users  19 2014-03-13 10:19 thread2.gcno ->
../lib/thread2.gcno
```

The gcno files are symbolic links to all the gcno files generated during gcov instrumentation of your source files.
The gcda files are result of the tool to convert standard output dumps to gcov result files.
Src is a link to the path of sources so that the relative path to the sources is the same than when preprocessing the source files with gcov.

3.3.2.4 Invoking the tool

The tool is invoked by the command <pathtodeployment>/ `COTS/gcovr-3.1/scripts/gcovr` when the currect directory is the one containing all the runX directories.

It will walk the directory tree to generate the synthesis of coverage.

The result is displayed as follows :

```
./COTS/gcovr-3.1/scripts/gcovr # Generation of the report
(WARNING) GCOV produced the following errors processing
/home/jdupeyro/lib/thread1.gcno:
           /home/jdupeyro/lib/thread1.gcno:cannot open graph file
           /home/jdupeyro/lib/thread1.gcno:cannot open graph file
           /home/jdupeyro/lib/thread1.gcno:cannot open graph file
           /home/jdupeyro/lib/thread1.gcno:cannot open graph file
        (gcovr could not infer a working directory that resolved it.)
(WARNING) GCOV produced the following errors processing
/home/jdupeyro/lib/thread2.gcno:
           /home/jdupeyro/lib/thread2.gcno:cannot open graph file
           /home/jdupeyro/lib/thread2.gcno:cannot open graph file
           /home/jdupeyro/lib/thread2.gcno:cannot open graph file
           /home/jdupeyro/lib/thread2.gcno:cannot open graph file
        (gcovr could not infer a working directory that resolved it.)
(WARNING) GCOV produced the following errors processing
/home/jdupeyro/lib/multithread.gcno:
           /home/jdupeyro/lib/multithread.gcno:cannot open graph file
           /home/jdupeyro/lib/multithread.gcno:cannot open graph file
           /home/jdupeyro/lib/multithread.gcno:cannot open graph file
           /home/jdupeyro/lib/multithread.gcno:cannot open graph file
        (gcovr could not infer a working directory that resolved it.)
------------------------------------------------------------------------------
File                                       Lines    Exec  Cover   Missing
------------------------------------------------------------------------------
```

```
/home/jdupeyro/gcovtest/src/multithread.c
                                          18      14    77%   19-20,28-29
/home/jdupeyro/gcovtest/src/thread1.c      5       5   100%
/home/jdupeyro/gcovtest/src/thread2.c      5       5   100%
------------------------------------------------------------------------------
TOTAL                                     28      24    85%
------------------------------------------------------------------------------
```

The warning messages cannot open graph file can be ignored.

The result displayed shows for each source file, the coverage ratio and the lines ranges not activated if not 100% and make a global value summary.