Valgrind

Contents

Valgrind

Stable: 09.10.2019 - 15:31 / Revision: 05.09.2019 - 09:30

Contents

1 Article purpose

This article provides the basic information needed to start using the Linux application tool: **valgrind**[1].

2 Introduction

The following table provides a brief description of the tool, as well as its availability depending on the software packages:

: this tool is either present (ready to use or to be activated), or can be integrated and activated on the software package.

: this tool is not present and cannot be integrated, or it is present but cannot be activated on the software package.

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		valgrind[1] is an instrumentation framework for buildin						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
valgrind	Monitoring tools	g dynamic analysis tools. Some Valgrind tools can automatically detect many memory management and threading bugs, and profile your programs in detail. This is tool for Linux application analysis.	✓	✓	✓	✗	✗	✓

3 Installing the trace and debug tool on your target board

3.1 Using the STM32MPU Embedded Software distribution

valgrind is installed by default and ready to be used with all STM32MPU Embedded Software Packages.

```
Board $> which valgrind
/usr/bin/valgrind
```

valgrind is integrated in weston image distribution through openembedded-core package: *openembedded-core/meta/recipes-core/packagegroups/packagegroup-core-tools-profile.bb*.

```
VALGRIND = "valgrind"
...
RDEPENDS_${PN} = "\
   ${PROFILETOOLS} \
   ${LTTNGUST} \
   ${LTTNGTOOLS} \
   ${LTTNGMODULES} \
   ${BABELTRACE} \
   ${SYSTEMTAP} \
   ${VALGRIND} \
   "
```

3.2 Using the STM32MPU Embedded Software distribution for Android™

3.2.1 Distribution Package

valgrind source code module is available Distribution Package in *external/valgrind*.

- To compile it (ensure the build environment is correctly set):

```
PC $> cd $ANDROID_BUILD_TOP
PC $> mma valgrind
```

- Check valgrind binary is available in system image:

```
PC $> ls out/target/product/<BoardId>/system/bin/valgrind
```

- Push the binary and dependencies (libraries) to the remote target file system:

```
# Remount first the target file system with write access
PC $> adb root; adb remount
PC $> adb push out/target/product/<BoardId>/system/bin/valgrind /system/bin/
PC $> adb shell mkdir /system/lib/valgrind
PC $> adb push out/target/product/<BoardId>/system/lib/valgrind/* /system/lib/valgrind/
PC $> adb shell sync
```

4 Getting started

Valgrind is designed to be as non-intrusive as possible. It works directly with existing executables/applications. It does not require to recompile, relink, or otherwise modify the program to be checked.

Here is a simple way to invoke valgrind:

```
Board $> valgrind [valgrind-options] <Program> [Prog_Args]
```

valgrind can be started without [valgrind-options]. More options can be turned on afterward, according the proposals provided in valgrind first-pass result.

Analysis is done during all phase of the program execution: program start, program run and program stop.

Below information is related to the Android™ distribution

Android™ also propose a web page [2] about valgrind usage.

Especially for using valgrind on an Android™ application.

5 To go further

5.1 Example

- Example of program analysis without any issue detected:

```
Board $> valgrind /usr/local/bin/fooProg
 ==351== Memcheck, a memory error detector
 ==351== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
 ==351== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
 ==351== Command: /usr/local/bin/fooProg
 ==351==
 push_led application example started...
 Monitoring line 13 on /dev/gpiochip0
 ^C...push_led application example exit.
```

```
==351==
==351== HEAP SUMMARY:
==351==     in use at exit: 0 bytes in 0 blocks
==351==   total heap usage: 2 allocs, 2 frees, 4,116 bytes allocated
==351==
==351== All heap blocks were freed -- no leaks are possible
==351==
==351== For counts of detected and suppressed errors, rerun with: -v
==351== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 3)
```

- Example of program analysis with memory leak issue detected (i.e. missing memory-freeing):

```
Board $> valgrind /usr/local/bin/fooProg
 ==360== Memcheck, a memory error detector
 ==360== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
 ==360== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
 ==360== Command: /usr/local/bin/fooProg
 ==360==
 push_led application example started...
 Monitoring line 13 on /dev/gpiochip0
 ^C...push_led application example exit.
 ==360==
 ==360== HEAP SUMMARY:
 ==360==     in use at exit: 20 bytes in 1 blocks
 ==360==   total heap usage: 2 allocs, 1 frees, 4,116 bytes allocated
 ==360==
 ==360== LEAK SUMMARY:
 ==360==    definitely lost: 0 bytes in 0 blocks
 ==360==    indirectly lost: 0 bytes in 0 blocks
 ==360==      possibly lost: 0 bytes in 0 blocks
 ==360==    still reachable: 20 bytes in 1 blocks
 ==360==         suppressed: 0 bytes in 0 blocks
 ==360== Rerun with --leak-check=full to see details of leaked memory
 ==360==
 ==360== For counts of detected and suppressed errors, rerun with: -v
 ==360== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 3)
```

The text highlighted in green in the above valgrind report suggests to turn on more options in order to focus on the reported issue.

6 References

- [1.01.1] http://valgrind.org/
- https://source.android.com/devices/tech/debug/valgrind

- Useful external links

Document link	Document Type	Description
Quick start	User guide	Official site
Wikipedia	Wiki	Documentation from Wikipedia

eval,disco (Generic term used, to complete configuration modules paths depending on used board)

Category:Monitoring tools

This category groups together all articles and subcategories related to the **monitoring tools and methods** that allow to get static status about the Linux® frameworks, the U-Boot or the TF-A.

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Trusted Firmware for Arm Cortex-A

Subcategories

This category has only the following subcategory.

1

- Linux monitoring tools (16 P)