TEC | Tecnológico de Costa Rica

High Performance Embedded Systems

Homework 1

Lecturer:
MSc. Mauricio Caamaño Bedoya

2019

PART I

Motivation

Cache misses and memory leaks are a common issue on software development, they are barely noticed when working on personal computers however when running a leaky program on an embedded system thing usually go bad. Common embedded systems have a limited amount of memory and a correct usage is critical to avoid the system to crash on the first 5min of operations.

Objective

With this assignment, the student will start experimenting with common memory analysis tools and will understand how to effectively track them within a running code.

Description

This homework is divided in two separated stages:

1. Basic Valgrind usage
2. LD_PRELOAD usage for memory leak detection

Valgrind usage (30 %)

Valgrind (http://valgrind.org/) is a poweful dynamic analysis tool with support on a huge range of platforms including x86 and ARM. One of the most common usages of Valgrind in embedded systems is to detect and debug memory leaks on complex programs.

For this part, the student will compile a source file (case1, case2 and case3). Later will use a Valgrind tool for analyzing their results.

Procedure

1. Install Valgrind on your Linux PC

   ```
   sudo apt-get install valgrind
   ```

2. Open the terminal and go to your working directory
3. Compile source file using

   ```
   gcc -Wall -pedantic -g name.c
   ```

4. Run the following command

   ```
   valgrind ./case1
   ```

5. Copy the output from the console to a text file named *Valgrind_case1.txt* for each binary.

6. The binaries file will automatically create a *log_case1.txt* for each binary file on its directory.
7. Store the *valgrind.txt* to a safe place so you can send them when delivering the project. (**NOTE:** log.txt file is really small - just 8-bytes long - and cannot be opened on a text editor).
8. Interpret the Valgrind log, analyze and explain the results using for example a source file. Save this result on a *case1.txt, case2.txt and case3.txt* file and put it among the other log files.

Recommended links:

http://valgrind.org/docs/manual/manual.html

http://www.linuxprogrammingblog.com/usingvalgrindtodebugmemoryleaks

Deliverables

The first part of the deliverable includes:

- *Valgrind_case1.txt*
- *Valgrind_case2.txt*
- *Valgrind_case3.txt*
- *case1.txt*
- *case2.txt*
- *case3.txt*

Using LD_PRELOAD to trace memory leaks (70 %)

LD_PRELOAD is an environment variable that is part of the Linux dynamic linker (http://man7.org/linux/man-pages/man8/ld.so.8.html) it can be used to specify the system to look at an specific set of ELF files (commonly shared libraries) before looking at the default ones.

LD_PRELOAD is commonly used to intercept the execution of specific calls in order to execute customized code instead (see http://www.goldsborough.me/c/low-level/kernel/2016/08/29/16-48-53-the_-ld_preload-_trick/). This is useful to track the memory allocation for specific or customized allocation calls that may not be standard (some systems uses proprietary memory allocation routines).

The objective of this section is to create a C-based tool that helps the developer to determinate whether a C-based application has memory leaks. The general syntax for the tool (which is going to be named *memcheck*) is as follows:

```
./memcheck [OPTIONS]
```

Where options are:

-a displays the information of the author of the program

-h displays the usage message to let the user know how to execute the application.
-p *PROGRAM* specifies the path to the program binary that will be analyzed, for example:

```
./memcheck -p /home/myuser/case1
```

Recommended link:

http://linux.die.net/man/3/fork

Note:

The binary *debug* has been provided as a helper program to debug your code. It allocs 20 times and frees just 15 so your leak count should be 5.

Implementation requirements

1. The application **MUST** use autotools for compilation.

Recommended links:

https://www.gnu.org/software/automake/manual/html_node/AutotoolsIntroduction.html

http://markuskimius.wikidot.com/programming:tut:autotools/

The layout of the tool should look similar to this:

```
memcheck/
├── configure.ac
├── lib
│   ├── libmemcheck.c
│   └── Makefile.am
├── Makefile.am
└── src
    ├── Makefile.am
    └── memcheck.c
```

libmemcheck.c is the code of the dynamic library that will be created while **memcheck.c** is the name of the tool source code.

2. The tool **MUST** accept at least the 3 options discussed previously (-v, -a, -p) and **MUST** implement them using *getopt.*

Recommended links:

https://www.gnu.org/software/libc/manual/html_node/Getopt.html#Getopt

3. The tool should be able to intercept the calls to *malloc* and *free* methods, keep track of how many times they are used on the application under analysis and provide a command line report as follows: ./memcheck ./case4

```
Analysis finished!
Memory allocations: 45
Memory free: 10
Total memory leaks found: 35
```

Deliverable

The deliverable for this stage is the source code for the tool: *memcheck*. Use evaluation criteria table for easy reference on guidelines.

Evaluation criteria

Item	Description	Percentage
Valgrind report	All the logs are provided and the data is correct.	30%
Memcheck autotools usage	Correct usage of autotools following the basic layout proposed. Make sure the configure.ac checks for the right requirements for the application (if any). Only the required autotools files are provided.	10%
Memcheck getopt implementation	Correct usage of getopt for the command lines options	10%
Memcheck library generation and usage (libmemcheck.so)	Correct build of libmemcheck.so and correct usage within memcheck application. (**Hint:** DO NOT use LD_PRELOAD in the command line)	10%
Memcheck implementation	Adequate implementation of the memcheck program. (**Hint:** DO NOT use the *system* call)	20%
Memcheck Functionality	Running the memcheck program with the buggy binary provides the correct data.	20%

Delivery date

Sunday, July 28th 12:00 m.n, and compact the deliverables using tar.gz or different format and upload in Tec-Digital.