TEC | Tecnológico de Costa Rica

LECTURE: MP-6171 SISTEMAS EMPOTRADOS DE ALTO DESEMPEÑO

Homework 1:
Valgrind Usage and Trace Memory Leaks

Lecturers:
MSc. José Araya Martínez
MSc. Sergio Arriola-Valverde

Second term, 2021

Contents

1 Introduction

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 5 min of operations. 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 [1].

1.1 Administrative Matters

Before we get down to business, some administrative affairs are to be discussed:

1.1.1 Team Formation

Work teams of 2 students should be made up and kept for all group assignments. Please send as soon as possible a list with the team members and their respective Email (the one associated with GIT) to the following Email Address: moiarayam@gmail.com

1.1.2 Forum and Communication

This project will be evaluated remotely, for this reason having a suitable online platform turns out to be very important to facilitate the communication between students and lecturers.

In order to do so, we will adopt a "community" approach by means of a **forum** in the **TecDigital** platform. In the forum, all students can create new topics as questions arise. All discussions are public to all the course members so that any fellow student can answer and/or add more information to the proposed question. Please avoid sending project-related queries directly to the lecturers' Email, as this prevents other students to benefit from the answer. In the forum we all are a team! The only restriction is to not share working source code in the forum, instead, we can create discussion and propose ideas and concepts that lead to the solution.

1.1.3 Plagiarism

Any evidence of plagiarism will be thoroughly investigated. If it is corroborated, the team will receive a **zero** grade in the project and the incident will be communicated to the corresponding institutional authorities for further processing.

1.2 Development Environment Overview

In relation to the development environment you can use at least two different approaches such as:

1. Use the same VM configuration used for Project 1. (It is highly recommended for this homework).
2. You can use your PC using a Ubuntu version under 18.04 LTS (In this case you must solve all dependencies in order to accomplish the application functionality).

1.2.1 Nomenclature

As we will work with different command-line consoles during the setup of the development environment, it is necessary to specify where the commands are to be executed. Table 1 summarizes the prompt symbols for the different command-line consoles.

Table 1: Prompt symbols for the multiple command-line consoles during the project.

Prompt symbol	Description
$	Host Linux PC
=>	Target: U-Boot
@	Target: Linux

2 Description

This homework is divided in two separated stages:

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

2.1 Valgrind Usage

Valgrind 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 the source files (case1, case2 and case3) attached to the homework folder or available here. Afterwards the student will use a Valgrind tool to analyze the results.

This part will explain the procedure to compile and execute files. Please follow the next instructions:

1. In your GitHub respository create a new folder called **Homerwork_1**.
2. Download and extract this file called **Valgrind Usage** into your new branch called **Homework_1**. Your layer repository should look like this:

```
master  ---- Homework_1
                    |--- Valgrind Usage
                            |--- Source Files
                                    |--- Case 1
                                    |--- Case 2
                                    |--- Case 3
                            |--- Report
```

3. To execute valgrind you must install it in your VM or own PC. To install valgrind, type in your terminal the following instruction:

```
# To install valdrind in your VM or OWN PC
sudo apt-get install valgrind

# For update packages use:
sudo apt-get update
```

4. In order to compile the source files please look for the respective folder (*e.g* Case 1) and use the following compile instructions.

```
# For case 1
gcc -Wall -pedantic -g -o case1 case1.c

# For case 2
gcc -Wall -pedantic -g -o case2 case2.c

# For case 3
gcc -Wall -pedantic -g -o case3 case3.c
```

5. When the binary files are already compiled please execute them using the following instructions.

```
# For case 1
valgrind ./case1
```

```
# For case 2
valgrind ./case2

# For case 3
valgrind ./case3

```

6. Copy the output from the console to a text file (*e.g* Outputcase1.txt) for each binary and into the folder called **Report**.
7. Interpret analyze and explain the results of the Valgrind report using for example the source file. Save this result (*e.g* Reportcase1.txt) in the folder folder called **Report**. More information about valgrind here and here.

2.2 LD_PRELOAD to trace memory leaks

LD_PRELOAD is an environment variable that is part of the Linux dynamic linker, it can be used to specify 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 here). 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 (using your OWN library) that helps the developer to determinate whether a C-based application has memory leaks (binary file is case4).

This part will explain the requirements in order to develop your custom memory leaks application:

1. In your branch in Github, please add the folder **Memory leaks** and look if the binary file called **case4** is contained. At this point your layer repository should look like this:

```
master  ---- Homework_1
                    |--- Valgrind Usage
                            |--- Source Files
                                    |--- Case 1
                                    |--- Case 2
                                    |--- Case 3
                            |--- Report
                    |--- Memory leaks
```

```
                                |--- case4
                                |--- Report
```

2. The memcheck implementation **MUST** use autotools for compilation. More information about autotools here and here. Due to the fact that the compilation must use autotools your repository should look like this now:

```
master  ---- Homework_1
                        |--- Valgrind Usage
                        |       |--- Source Files
                        |       |       |--- Case 1
                        |       |       |--- Case 2
                        |       |       |--- Case 3
                        |       |--- Report
                        |--- Memory leaks
                                |--- memcheck
                                |       |--- configure.ac
                                |       |--- lib
                                |       |       |--- libmemcheck.c
                                |       |       |--- Makefile.am
                                |       |--- Makefile.am
                                |       |--- src
                                |               |--- Makefile.am
                                |               |--- memcheck.c
                                |--- case4
                                |--- Report
```

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

3. The tool MUST accept at least the 3 options discussed previously (-h, -a, -p) and **MUST** implement them using getopt. More information to create fork here.

```
Usage
./memcheck [ -p ./PROGRAM ] [-h] [-a]
-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 path/case4

NOTE: Before executing case4 you need to give permissions under read and write use:
```

```
sudo chmod +777 case4
```

4. 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 -p case4, for example your visualization must look like this:

```
# Output example

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

3 Deliverable and Grading

This section explains the work flow requirements, deliverable and grading related to this homework. Please read and make sure to organize your project execution and deliverable over work time and not just before the deadline.

The grading of this project will be based on:

1. Little report per case in format txt.
2. Git Repository

3.1 Work flow requirements

In relation to the Git repository, several aspects are to be considered. It is important to follow them in order to organize your code development and prepare your repository:

1. Open a Git account on a free Git hosting (GitHub)
2. Create a Git repository named as follows: $< group\# >$ _$HPEC$_2021. Where the $group\#$ shall be the assigned group number. For example, for Group 1, the name of his Git repository shall be: $group1$_$HPEC$_2021

3. If the repository is private then provide access to the user moiarayam@gmail.com (GitHub) and make sure to provide write permissions.
4. The Git repository shall contain two main branches: master and develop
5. Initially the develop branch is created from the master.
6. When working on a project, the student shall create a new working branch from develop (e.g. ld_preload or valgrind_usage) and when the feature is ready the branch must be merged to develop. Any additional fix or modification after merge must require the process to be repeated (i.e. create the branch from develop and merge the changes back).
7. Once the code in develop branch is ready to be "released" it shall be merge to master and a tag must be created. The process is described in Figure 1. You can find more information in the git tutorial and here.

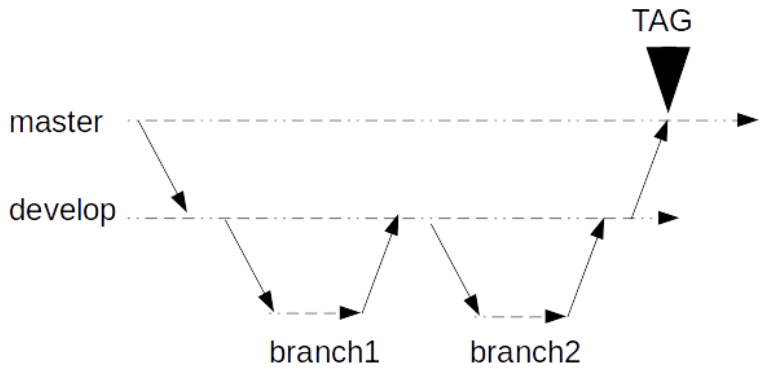

Figure 1: Repository structure for assignments or projects

3.2 Folder Structure of your Deliverable

In relation to deliverable documentation and folder structure you **MUST** follow the folder organization in your $user\_id$.tar.gz (*e.g* $group1\_HPEC\_2021$) and Git repository as depicted below:

```
master  ---- Homework_1
                |--- Valgrind Usage
                |       |--- Source Files
                |       |       |--- Case 1
                |       |       |--- Case 2
                |       |       |--- Case 3
                |       |--- Report
                |--- Memory leaks
```

```
|--- memcheck
|       |--- configure.ac
|       |--- lib
|       |      |--- libmemcheck.c
|       |      |--- Makefile.am
|       |--- Makefile.am
|       |--- src
|              |--- Makefile.am
|              |--- memcheck.c
|--- case4
|--- Report

```

3.3 Grading

The grading of this homework will be based on Table 2.

3.4 Deliverable Submission

To submit your deliverable you have to:

1. Before sending your documentation and deliverable please follow the folder structure established on section 3.2.
2. Compress your deliverable folder in zip format and look in TEC-Digital for the delivery section for Homework 1.
3. Delivery date is projected for Sunday June 6^{th} 2021, deadline 23:45 afterwards, the link access in TEC-Digital will be closed.

References

[1] M.Madrigal. I loss my memory, 2016. [Online; accessed 15-June-2020].

Table 2: Homework evaluation criteria

Item	Description	Percentage
Valgrind report	In-depth report analysis and discussion based on Source files and results	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 getop implementation	Correct usage of getop 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	Adecuate implementation of the memcheck program. (**HINT:** DO NOT use the system call)	20 %
Memchechk functionality	Running the memcheck program with the buggy binary provides the correct data.	20 %
Total		**100 %**