M1522.000800 System Programming, Spring 2017
Linker Lab: Memtrace
Assigned: Wednesday, April 5, Due: Thursday, April 13, 23:39

Introduction

In this lab, we use the at-load-time library interpositioning to trace calls to the dynamic memory management system of a compiled binary program. Your memory tracer will be able to log all dynamic memory management calls of a program, and identify memory blocks that a program does not free. We then use a low-level libraries to identify the call sites that allocate the unfreed memory.

If you are still not satisfied, you can augment your memory tracer to prevent incorrect memory deallocations of random binaries in an optional bonus question.

By solving this lab, you will learn a lot about library interpositioning and simple management of data within a shared library. A lot of the code is provided already, this lab will also force you (an teach you how) to read someone else's code. We will also make use of a debugger library that will show you how to pinpoint exact program points in your running programs.

The lab may require a significant effort if you are not familiar with C programming. We provide enough hints and support in the lab sessions so that everyone can solve the lab with a good mark. Nevertheless, start early and ask early!

Logistics

Installation

Obtain the *linklab-part1.tar* tarball from

```
/home/handout/ folder in sysprog1.snucse.org or sysprog2.snucse.org
```

and unpack it into a directory of your choice.

```
studentx@Sysprog2017Spring ~ $ tar xvzf linklab.tgz
studentx@Sysprog2017Spring ~ $ ls -l
```

Compiling, Running and Testing

`Makefiles` in the various directories assist you with submitting your compiling, running, and testing your implementations. Do not modify the Makefiles unless explicitly instructed. Run make help to find out which commands the Makefile support.

A set of test programs is provided with which you can test your solution. Note that we will use a different test set to evaluate your submission..

```
studentx@Sysprog2017Spring ~/linklab/part1 $ make help
```

Make sure to compile the test programs before running *make run test*

```
studentx@Sysprog2017Spring ~/linklab/part1 $ cd ../test
studentx@Sysprog2017Spring ~/linklab/test $ make
```

Submission

To submit your solution run `make submit` in the `top` directory. You must fill in your student ID and name in the file STUDENT.ID or the submission will fail. You can submit as many times as you want; the very last submission will be graded and determines the submission date.

```
studentx@Sysprog2017Spring ~/linklab $ make submit
```

You will see the message "Submission successful." if your submission has been received correctly by the server. If the submission to the server fails, you can email a tarball of your submission by email to
jichoi@dcslab.snu.ac.kr.

Dynamic Memory Management

Overview

In many languages, memory is explicitly allocated and sometimes even deallocated by the programmer. The POSIX standard defines the following four dynamic memory management functions that we want to intercept.

`void *malloc(size_t size)`

`malloc` allocates `size` bytes of memory on the process' heap and returns a pointer to it that can

subsequently be used by the process to hold up to `size` bytes. The contents of the memory are undefined.

`void *calloc(size_t nmemb, size_t size)`

`calloc` allocates `nmemb*size` bytes of memory on the process' heap and returns a pointer to it that can subsequently be used by the process to hold up to `size` bytes. The contents of the memory are set to zero.

`void *realloc(void *ptr, size_t size)`

`realloc` changes the size of the memory block pointed to by `ptr` to `size` bytes. The contents are copied up to `min(size, old size)`, the rest is undefined.

`void free(void *ptr)`

`free` explicitly frees a previously allocated block of memory.

Use the Unix manual to learn more about `malloc`, `calloc`, `realloc`, and `free`:

```
studentx@Sysprog2017Spring ~/linklab $ man malloc
```

Dynamic memory management functions are defined in the standard library. Include `stdlib.h` in your program to have access to `malloc`, `calloc`, `realloc`, and `free`.

Example

The following example program demonstrates how dynamic memory management functions can be used.

```c
#include <stdlib.h>

void main(void) {
  void *p;
  char *str;
  int  *A;

  // allocated 1024 bytes of memory
  p = malloc(1024);

  // allocated an integer array with 500 integer
  A = (int*)calloc(500, sizeof(int));

  // allocate a string with 16 characters…
  str = (char*)malloc(16*sizeof(char));

  // ...then resize that string to hold 512 characters
  str = (char*)realloc(str, 512*sizeof(char));

  // finally, free all allocated memory
  free(p);
  free(A);
  free(str);
}
```

example1.c

Note that all allocators return an untyped pointer (`void*`) that needs to be converted to the correct type in order to prevent compiler warnings.

Part 1: Tracing Dynamic Memory Allocation (20 Points)

Subdirectory: part1/

In this part, your job is to trace all dynamic memory allocations/deallocations of a program. Print out the name, the arguments, and the return value of each call to `malloc`, `calloc`, `realloc`, and `free`. When the program ends, print statistics about memory allocation (number of bytes allocated over the course of the entire program, average size of allocation).

For the program `test1.c` given below

```
#include <stdlib.h>

void main(void) {
  void *a;

  a = malloc(1024);
  a = malloc(32);
  free(malloc(1));
  free(a);
}
```

test1.c

the following output should be generated (the pointer values may differ from system to system):

```
studentx@Sysprog2017Spring ~/linklab/part1 $ make run test1
[0001] Memory tracer started.
[0002]           (nil) : malloc( 1024 ) = 0xb87010
[0003]           (nil) : malloc( 32 ) = 0xb87420
[0004]           (nil) : malloc( 1 ) = 0xb87450
[0005]           (nil) : free( 0xb87450 )
[0006]           (nil) : free( 0xb87420 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          0
[0012]
[0013] Memory tracer stopped.
studentx@Sysprog2017Spring ~/linklab/part1 $
```

Leave the statistics on the total number of freed memory bytes (`freed_total`) at 0 for now.

You can start from scratch or implement your solution by extending the skeleton provided in `~/linklab/part1/memtrace.c`. You do not need to modify `callinfo.c/h` for this part.

Use the logging facilities provided in `~/linklab/utils/memlog.c/h`. This will ensure that your output looks exactly as above and we can automatically test your submission for correctness.