Verkada Programming Assignment

confidential, please do not share. Please read the whole thing! FAQ at the bottom.

Challenge

Your goal is to create a glibc-based shared library object (.so) that wraps libc's malloc(), realloc(), calloc(), and free() (see: *man 3 malloc)* and instruments them to provide statistics on demand.

The wrapping should utilize the LD_PRELOAD environment variable technique so that it can be applied to any arbitrary dynamic binary. Initialization could be performed on the first call to any of the wrapped functions.

Every 5 seconds or so print out the statistics summary to stderr followed by two newlines. You don't have to use another thread; it is acceptable to just check the time since the last print on an invocation to malloc.

Deliverable

tar file with:

- .c / .cc file (C/C++)
- Makefile (GNU make, utilizing only gnu compiler tools)

Notes

You are not expected to make this particularly fast. It's geared toward more of a memory profiling/debugging tool. Make sure to document assumptions and include any test code you created during the process.

You should track both the overall allocations and current allocations, where current allocations is restricted to pointers not yet freed or realloc'd and overall allocations includes every pointer returned since execution started.

You can use any information from the web as reference. However, please do not wholesale cut and paste anything.The goal should be to create a production grade solution. Ideally it should be thread-safe (pthreads), allowing it to be used with threaded daemons, for example.

Hint: you will probably need to do something to allow your malloc to be called recursively. A good test is to use coreutils find on a big directory for testing.

FAQ

1. The project is taking me more than 4 hours to complete, am I slow?

Answer: You're fine. This task can be complex and depending on design decisions, can take more or less time.

2. Does the resulting stats need to display according to the memory requested by the executable or the actual memory allocated by malloc (or equivalent)? The reason being malloc may allocate larger memory areas to avoid alignment issues.

Answer: You should use the memory requested - that's what we're expecting. Doing it with actual allocated bytes as well could be a bonus.

3. Should I add a readme?

Answer: Yes.

4. Should I add test files or test instructions?

Answer: Yes

5. Should I include test code I created during the process?

Answer: Yes

Example

Format the output however you like, but if you need inspiration, here's an example:

```
>>>>>>>>>>>>> Sat Jan  6 00:36:26 UTC 2018 <<<<<<<<<<<
Overall stats:
6,123,231 Overall allocations since start
84.3MiB Current total allocated size

Current allocations by size: ( # = 8,123 current allocations)
0 - 4 bytes: ##########
4 - 8 bytes:
8 - 16 bytes: ####
16 - 32 bytes:
32 - 64 bytes:
64 - 128 bytes:
128 - 256 bytes:
256 - 512 bytes: ##
512 - 1024 bytes: #
1024 - 2048 bytes: #
2048 - 4096 bytes: #
4096 + bytes: #######

Current allocations by age: ( # = 8,123 current allocations)
< 1 sec: ###
< 10 sec: ##
< 100 sec: ##
< 1000 sec: ########################
> 1000 sec:
```

Non-Linux Users

If you don't have easy access to a linux machine, you can create the necessary environment with docker. Sample Dockerfile:

```
FROM ubuntu:18.04

RUN apt-get update && apt-get install -y build-essential software-properties-common
RUN add-apt-repository ppa:ubuntu-toolchain-r/test && apt update
RUN apt install -y gcc-9
```

```
RUN update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 60
```

Build and run, mounting the current directory into the container:

```
docker build . -t ubuntu-dev
docker run -it -v `pwd`:/home/ubuntu/dev ubuntu-dev:latest
```