Search this site...
Assignments
Write a shell
Memory allocation
System calls
C programming

Due @ 18:00 on **3 Mar 2023**

Implement your own version of `malloc(3)`, `free(3)` and related functions.

Having discussed memory allocation mechanisms and strategies in Lecture 5, it's time to build your own memory allocator! This assignment will help you:

- understand the challenges and trade-offs inherent in memory allocation,
- practice interacting with the OS kernel via system calls and
- practice implementing data structures in C.

Directions

You can choose whatever memory allocation algorithm you like: best-fit, first-fit, buddy, something else, etc. I won't require you to implement **all** of the `malloc(3)`-like functions, but I do want to see implementations of `malloc(3)`, `realloc(3)` and `free(3)`. In addition to functions that look like standard `libc` functions, you also need to implement some functions that can be used to control or query your allocation library; see the bottom of the rtos-alloc.h header file.

In addition to implementing source code, you must **evaluate its performance**. I will expect you to plot the time taken to perform a variety of allocations of different sizes against the time taken by libc's `malloc(3)` to perform the same allocations. You may wish to use `clock_gettime(2)` with the `CLOCK_PROCESS_CPUTIME_ID` argument (or see this helpful reference). The choice of what range of values should go on the X axis of the plot is left to you to investigate.

Materials

I'm providing you with a few header and source files to get started with. You only need the first header file (rtos-alloc.h), but you may find the others helpful as you work through the problem.

rtos-alloc.h
: This is the header file that declares the functions you need to implement.

test.cpp
: This is some test code that I've written for these functions. I don't claim that it's exhaustive (yet) and I reserve the right to add more tests between now and the due date, but this should get you going in the right direction when thinking about tests.

passthrough.c
passthrough-internal.h
passthrough-internal.c
: This is an implementation of the required functions that, instead of doing the actual work of allocation itself, simply passes calls through to `malloc(3)`, `free(3)`, etc. It's actually more complicated than that, as it keeps track of outstanding allocations in order to answer queries such as `rtos_alloc_size()`, but it's still not a real solution to the problem I asked you to address. As such, feel free to inspect it, use it in your test code and perhaps even pick up a few C idioms.

Deliverables

To complete this assignment you must submit both source code and your plot(s) to Gradescope. Use whatever vector or raster image format is convenient for you.