Toy Buddy Allocator

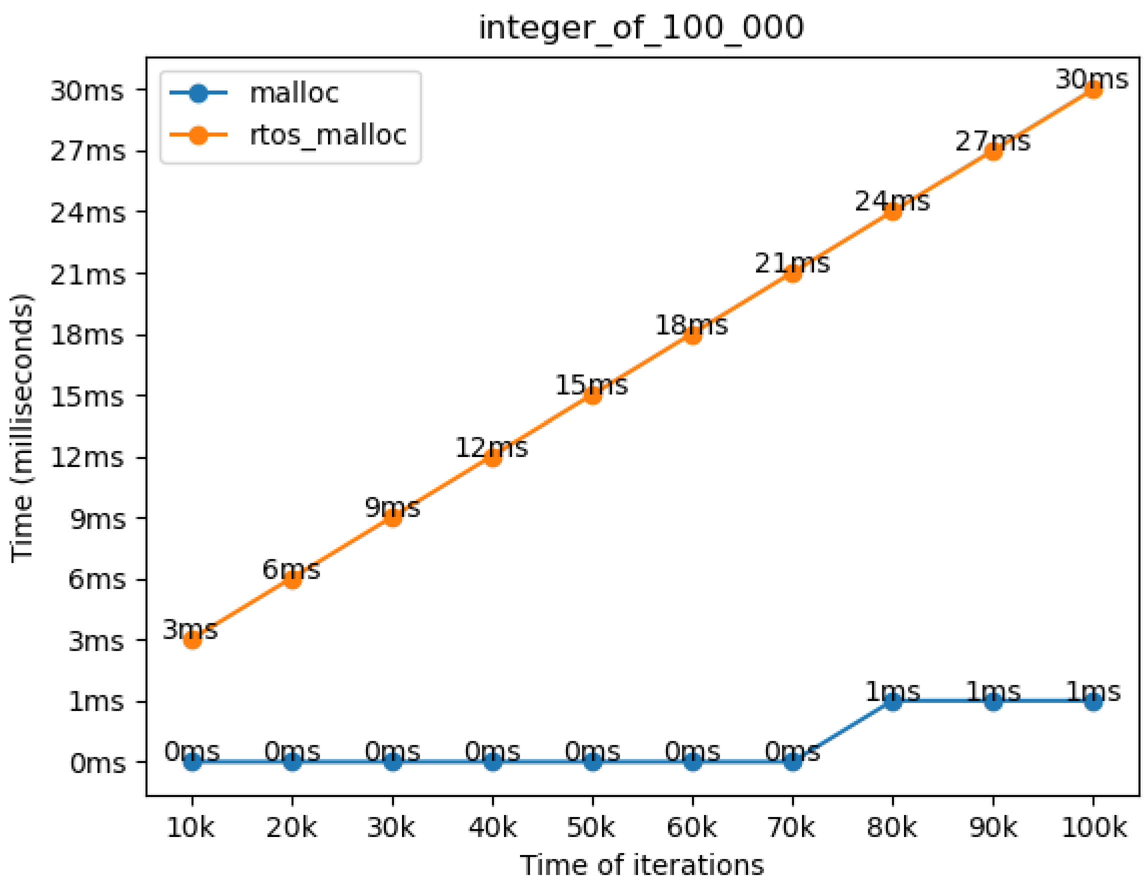
Student: Zhen Guan (202191382)

Ideas

- Allocate a huge stack memory as "physical memory"
- Total size of memory is 2^27 bytes (Don't know why it allowed me to allocate that many bytes on stack)
- Header (32 bytes) stored in the beginning of each block
- Runs without malloc
- Divide memory blocks by recursion

Performance evaluation

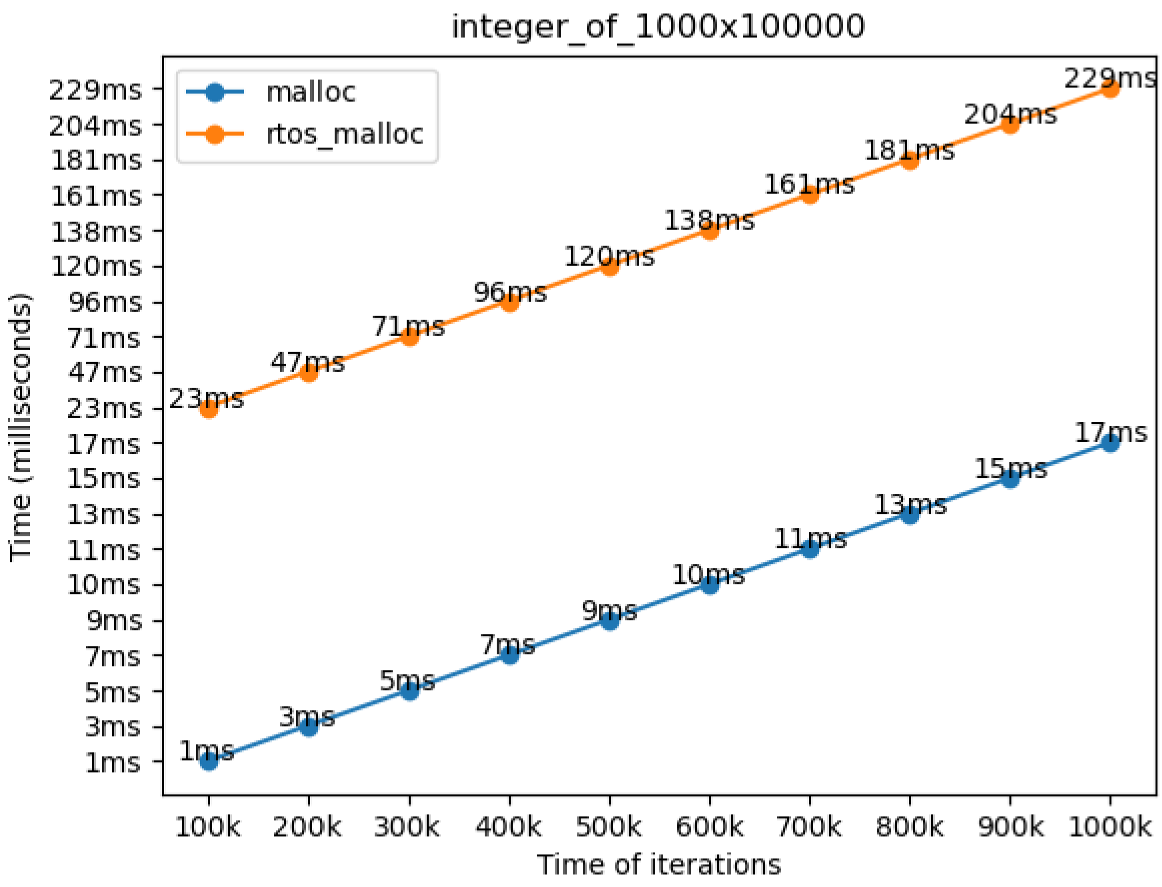
- Data collected by `std::chrono::high_resolution_clock`
- Running `rtos_malloc/rtos_free` and `malloc/free` on the same task with varying times of iterations
- Data visualized by Python pip `matplotlib`



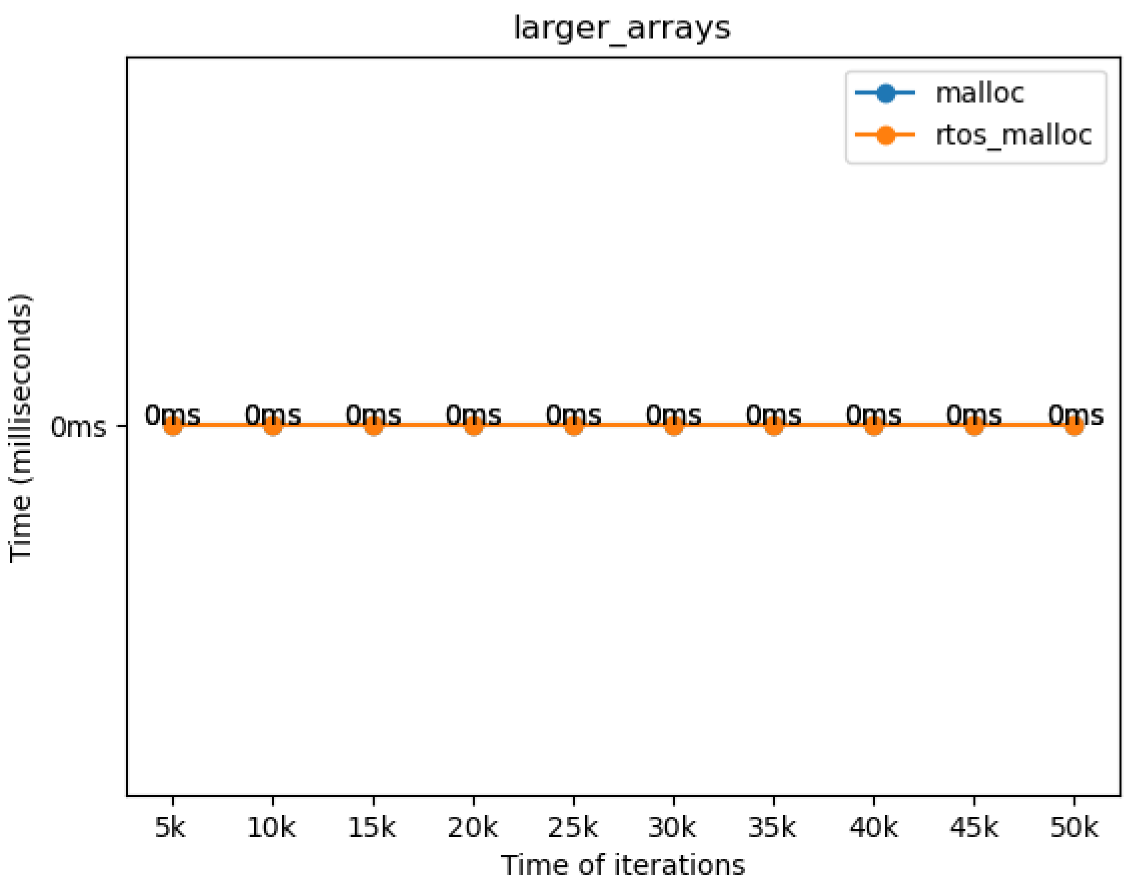
Above is the performance of allocating and freeing 100,000 integers. The x-axis is the number of iterations, and the y-axis is the time in milliseconds. The blue line is the time of `rtos_malloc` and `rtos_free`, and the orange line is the time of `malloc` and `free`.

With the increase of iterations, the time of `rtos_malloc` and `rtos_free` is also increasing linearly. This is as expcected. `malloc` from Linux kernel has very advanced memory management, and it is much faster than my implementation.

`0 ms` is actually microseconds lower than 1000.



Above is an extremely challenging test where both `rtos_malloc` and `malloc` had linear increasing time. However `malloc` is still much more advanced.



Regarding multiple medium size arrays, these two functions both finished in less than 1 ms.

What could be improved in `rtos_malloc`

- Use a flatten binary tree to manage the memory blocks.
- Learn more about memory management before writing codes.
- Use `brk/sbrk` or `mmap` instead of stack memory.