L1 — Important Issues in the Experiment

181240020 Hu Junhao

April 2, 2020

1 Detailes in the alloc and free design

1.1 Agreement

- The header is inserted within the heap and "right" before the beginning of the allocated memory. There might be some space between the header and the allocated memory due to the alignment requirement. And the address of the header is inserted right (without quotation mark) before the beginning of the allocated memory.
- Whenever we got an allocation request with n bytes, we always round it up to the nearest number of bytes which can be divided by 8. In this case, although we might waste tons of memory when small amount like 1 byte of memory is requested, we are assured that the address of each header is appropriate for data alignment needs.
- We did not use all of the heap memory. We delibrately cut out the last peice of memory for a void header.
- When we call alloc(0), we return NULL.

1.2 alloc

- Whenever we got an allocation request with n bytes, we always round it up to the nearest number of bytes which can be divided by 8. And at the same time, we store the address of the header right before the beginning of the allocated memory. This altogether takes O(1) time. But we have to spend extra time sweeping through the list tp find out which block is suitable for the allocated memory, which takes O(n), and n is the length of the list, which might grow very large as the call to free function increases.
- We have bigalloc and kalloc. Since we know that the biggest alloc call is 4096, we can bigalloc a big "page" beforehand and alloc the comparably small memory in the "page".

1.3 free

- When we want to free one block, we fisrt get an address of the to-be-freed memory block and take a step forward, get the address of the header, free this block and return it to the free list, which only takes O(1).
- Notice that when we free a certain address, we simply set the sptr of the header to 0 without returning it to the free list. After certain amout of time, for example, after certain amount of frees are called, we go through the whole allocted lists and find out those big "pages" whose members are all freed. And we return this whole "page" back to the free list.

2 Lock

2.1 First try

- Originally, we only put a large, heavy lock whenever we need to access the heap and the corresponding header — a list structure. This is helpful in the very beginnig, in that we can fully focuse on the correctness of the alloc and free function.

2.2 Modification

- Then we only add the heavy big lock when we call bigalloc. And when we allocate small memories we do not need any locks, which boosts the level of parallelism.

3 Problems encountered

- It is really hard to find the correct parameter, which optimize the whole allocating process. For example, how large a big "page" should be? Should it be 4096, or $4096*2^n$, or $4096*2^n+b$? We take the last type of parameter since we need to keep space for the headers inserted within the pages. When I set the page size to be $4096 * 2^n$, the programm fail on online judge. Nevertheless, $4096 * 2^n + b$ is much more faster the that and the programm passes.
- There is a challenging request — the restriction on the starting address of the allocted area. Considering the fact that we have incorporated headers right before the allocated space, we should be really careful about the size of the header. If the size of the header is 2^n, it should be much easier for us to implement the programm and we are also able to save a lot of space which might become internal fragment if the size of the header is, for example, an odd number.