Report - Project #1: Malloc Library

Name: Xichun Gou NetID: xg84

1 Overview

1.1 Introduction

Each region consists of two parts: metadata and data storage. Metadata, as a struct, contains region information: a pointer to the previous region, a pointer to the next region, and size (the size of the data storage part). Manage free regions through the data structure of the doubly linked list, as a freelist. When a region needs to be allocated, the region needs to be removed from the freelist. When a region is freed, the region needs to be inserted into the freelist.

```
struct metadata{
     struct metadata * prev;
     struct metadata * next;
     size_t size;
};
typedef struct metadata meta;
```

1.2 implementation of main functions

Main functions are implemented as follows:

ff_malloc	First fit malloc: ends the loop immediately when find the first fit
bf_malloc	Best fit malloc: get best fit allocation address by calling compare()
allocate	Called by ff_malloc and bf_malloc: When find the fit free region, call split_region; Otherwise, use sbrk()
split_region	If there is enough space, split the area according to the required size; if there is not enough space, remove the region directly from the freelist by calling remove_freelist

remove_freelist	when region is allocated, remove the region from the freelist
ff_free	First fit free
bf_free	Best fit free, same as first fit free
free_region	Called by ff_free and bf_free: Get the pointer pointing to metadata of the freed region, call insert_freelist
insert_freelist	when region is freed, insert the region to freelist in order and merge the adjacent free regions by calling merge_back and merge_front
merge_back	if adjacent to the back, merge current region and the next region
merge_front	if adjacent to the front, merge current region and the prev region

1.3 Details that need to be explained

1) reset size

When the size of the region is equal to the sum of sizeof(metadata) and the size to be allocated, the region cannot be further split in this case. Because if the region is split, the remaining part is too small to keep track of. It should be noted that at this time, in order to avoid errors, the size of the region needs to be updated to the size that needs to be allocated.

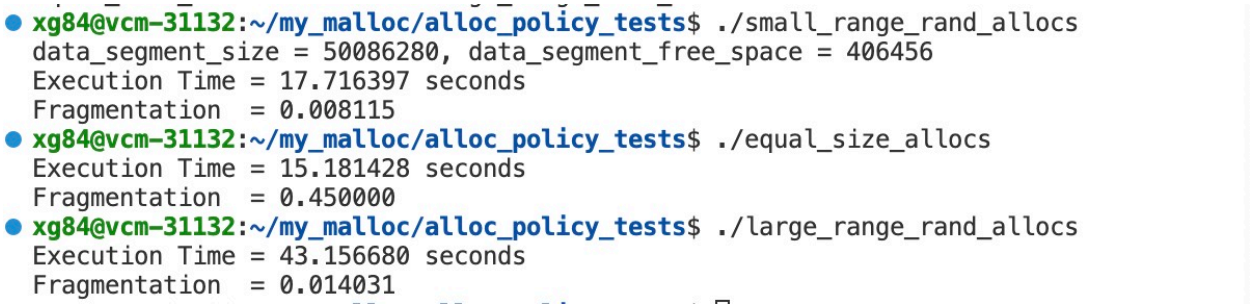
2) split the later part of the region

head

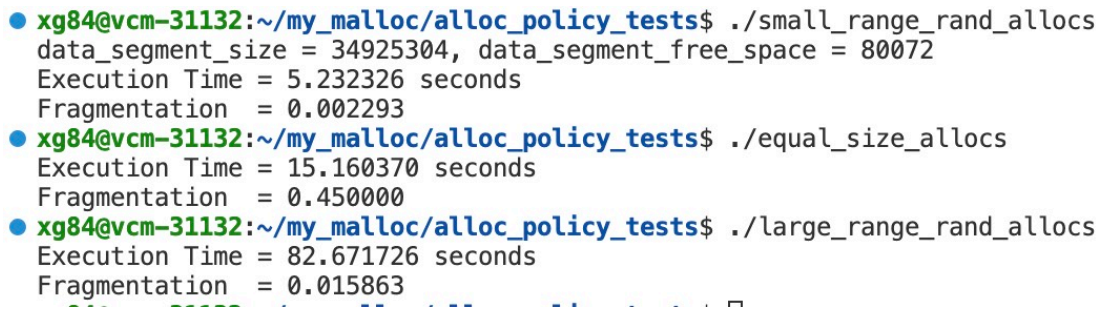
When the region is large enough to be split, the latter part of the region is split as the allocated part; while the front part remains in the freelist. Implemented in this way, you can avoid changing the direction of the pointer, and only need to update the size of the region in the metadata.

2 Results from performance experiments

2.1 First fit:



2.2 Best fit:



3 An analysis of the results

3.1 equal_size_allocs

The size of the region that needs to be allocated when calling malloc is fixed, so each region in the freelist is the same size. Therefore, whether it is ff or bf, the fit region found by malloc every time is the first region in the freelist. Therefore, the running time and fragmentation of the two are similar or the same, the time is about 15s, and the fragmentation is 0.45

3.2 small_range_rand_allocs

This program works with allocations of random size, ranging from 128 - 512 bytes (in 32B increments). Therefore, the size of the region in the freelist is

random. In this case, the bf strategy will outperform the ff strategy. Specifically, for example: the sizes that can be allocated to the regions in the current freelist are 30, 20, and 10 in sequence, and the order in which we call malloc is 10, 20, and 30.

At this time, for ff: the region of size 30 will be split and allocated. When the required allocation size is 30, the sbrk() function should be called. For bf, the above requirements can be met without calling the sbrk() function, so the time of ff will be longer than that of bf, which are about 17s and 5s respectively, and fragmentation is about 0.008 and 0.002 respectively.

3.3 large_range_rand_allocs

Similar to small_range_rand_allocs, the size of the region is random. The difference is that this program works with allocations of random size, ranging from 32 - 64K bytes, with a larger range. A large range may lead to a longer freelist, and the program will spend more time on loop search. For bf, it is necessary to search and compare each time, and select the best fit region, which will take more time than the ff strategy, and compared with calling the sbrk() function, this type of time may be even greater. Therefore, from the perspective of running time, ff is faster than bf, which are about 43s and 82s respectively.

4 conclusion

In reality, the size range of the data should be huge, so in this case, based on the experimental results, the ff strategy may be more appropriate. If the size range of the data is small, my bf strategy may be more appropriate.