ECE650-project1

- Author: Yang Xu
- netID: yx248
- Data: 01/24/2023
- Instructor: Rabih Younes
- Course: ECE651

The Allocation Policies Implementation:

For this project, we should implement the allocation policies. We should implement `6 methods` :

```
//First Fit malloc/free

void *ff_malloc(size_t size);

void ff_free(void *ptr);

//Best Fit malloc/free

void *bf_malloc(size_t size);

void bf_free(void *ptr);

unsigned long get_data_segment_size(); //in bytes

unsigned long get_data_segment_free_space_size(); //in byte
```

For the allocation policies, we should apply for enough heap space to store the required size space by `malloc method` . And free the specific address with `free method` .

The malloc needs to find an enough space region in the free space in the total heap space, and the addresses are ordered in the free region list, so I think the doubled linked list is a good way to store the free regions. If we need to find a required free region, we can traverse the linked list to find it. If we need to store a new free region in the list, we can add or merge or split it easily by the properties of linked lists.

The first fit methods:

ff_malloc

Firstly, check the free list, when finding the first free region which is enough for the required size with the metadata size, split or use the free region. If the region's left part is larger than the metadata size, split the size need to use for the required size; otherwise, use the total region space. Then, if no free region is enough, use the sbrk system call to increase the heap size for the required size, if not enough memory space can be increased for the heap, return NULL.

ff_free

Check the address need to be freed whether it is NULL, if it is NULL, return back. Find the correct position for the address which needs to be freed. The first address which is larger than the required address will be the required freed address next. And check whether the required freed address is continuous with its prev or next, and merge them together.

The best fit methods:

bf_malloc

Firstly, check all of the free list, check the closest enough size free region, and split or use the free region. If the region's left part is larger than the metadata size, split the size need to use for the required size; otherwise, use the total region space. Then, if no free region is enough, use the sbrk system call to increase the heap size for the required size, if not enough memory space can be increased for the heap, return NULL.

bf_free

Because the `bf_free()` is totally the same as the `ff_free()` , so I call the `ff_free()` to implement the `bf_free()`

Two performance study report methods:

get_data_segment_size

This function is for returning the totally heap size. I set a global variable to update the heap size when it uses the system call `sbrk()` function.

get_data_segment_free_space_size

This function is for returning the totally free region size in the heap. I set a global variable to update the free size when doing free operations or doing malloc operations for the free region.

Results from My Performance Experiments

FF	Running Time	Fragmentation	BF	Running Time	Fragmentation
ff_equal_size	22.52s	0.45	bf_equal_size	22.34s	0.45
ff_small_range_rand	7.27s	0.060	bf_small_range_rand	1.84s	0.022
ff_large_range_rand	55.07s	0.093	bf_large_range_rand	67.67s	0.042

Analysis of The Results

equal_size_allocs:

The `equal_size_allocs` tests' running time are almost the same, and the fragmentation are the same. I found that all data malloced into the heap are the same size which is `ALLOC_SIZE = 128`, and the data would be freed in order, so I think both the bf and ff way will use the same free region in the free list which is the first fit region in the free list. Therefore, both `equal_size_allocs`'s running time and fragmentation should be the same, but because of normally time erorr the time will be very close but not the same.

small_range_rand_allocs:

The `small_range_rand_allocs`'s results have a very large difference between first fit way and best fit way. For the fragmentation, the best fit way will choose the best-matched free region, and the first fit will choose the first free region, which may not be the best-matched one, so the first fit way's fragmentation will be larger than the best-fit way's. And the running time also has a great difference. For this case, there are many small range regions in the free list. The first fit way will use the first free region, which may not be the best-matched one, so it can do split operations, then there may not be enough space left for after requests, and the `sbrk()` function will be called, which is a system call function, so it will spend a lot of time. For example, the free list has five free regions, whose sizes in order are `[1,2,3,4,5]`, then try to malloc 5 regions to use whose size in order are `[5,4,3,2,1]`, then the first fit way will call two more times `sbkr()` than best fit way. What's more, the best fit way may reduce the length of the free list after each best match operation, which can reduce the times of traversing the free list, which can reduce the running time.

large_range_rand_allocs:

The `large_range_rand_allocs`'s results have a very large difference between first fit way and best fit way. For the fragmentation, the best fit way will choose the best-matched free region, and the first fit will choose the first free region, which may not be the best-matched one, so the first fit way's fragmentation will be larger than the best-fit way's. The running time are close but the best fit way was about 10s slower than first fit way. For this case, each free region is large, so almost malloc requirements need to do the split operation, whatever we use the best fit way or first way, so for the best fit way cannot reduce its free list like the `small_range_rand_allocs` case, so it will not reduce the time for each time free list traversal. Therefore, the running time for best fit way is slightly slower than the first fit way about 10s.