ITP 30002-02 Operating System, Spring 2020

Homework 5

Park Ye Gyeom/21600277/21600277@handong.edu

1. Introduction

This task is to implement Smalloc 2.0, an improved version of Smalloc 1.0, and related APIs. Of the total five tasks, task1 is to replace the existing first fit algorithm with the best fit algorithm that allocates the most suitable space among Unused space. The requirements of Task2 identify the prev and next status of the free area. If Unused, it is to combine the two spaces so that memory space can be used more efficiently. The requirement in Task3 is to implement print_mem_uses(), an API that outputs the state of memory currently used, and to implement this requires an understanding of how dsize and meta-data values are used. Task4 is the implementation of realloc, an API that changes the allocation size of space currently in Busy state. If the space you are trying to change is larger than the existing space and larger than the free space, the function should be implemented using memory copy. The requirement in Task5 is to implement sshrink() to reduce break point of heap. A negative value could be used in sbrk() syscall to implement sshrink().

2. Approach

2.1 Task 1

Task1 needs to be replaced with the Bestfit algorithm, so it can be divided into two cases.

If itr->dsize == size, perform break because it is the most ideal result.

If itr->dsize > size + sizeof(sm_container_t), find the hole with the smallest bestfit value and implement smalloc that works with bestfit algorithm.

2.2 Task 2

It maintains the function of existing sfree and additionally implements the function of merging containers in adjacent unused state.

If itr != sm_head.next && itr->prev->status ==Unused, merge itr and prev container with one container. And itr -> next != &sm_head && itr->status == Unused If this condition is also met, combine itr and next container into one container. Note that not only the dsize value but also the meta-data value should be added.

2.3 Task 3

The repetition statement distinguishes between busy and unused spaces in the heap. Then calculate the value of the space allocated to each state. The value of Total will be Busy + Unused. Each result is printed in the form of

"fprintf (stderr,total,busy,unused)."

2.4 Task 4

Implement slalloc, a function that changes the size of the memory space by the requested size.

- If itr->dsize == size, there is no change.

- If itr->dsize > size, divide the existing space through sm_container_split (itr, size) and free the remaining space.

- If itr->dsize < size, find unused space by loop statement and calculate the free value. If the free value can accommodate the size, perform itr->dsize += free and perform the sreallock(). If the free value is not sufficient, replace it with the existing itr->status = Unused and copy the data to the new location, hole, via memcpy (_data(hole),_data(itr),itr->dsize).

2.5 Task 5

Navigate in the direction of Sm_head->prev and perform a loop statement until you meet a container in Busy state. When performing a repeating statement, the heap space that is not currently used is identified and accumulated in the free. If the free value is calculated, pass the -free value to the sbrk() system call as the factor value to reduce the break point of the heap.

3. Evaluation

Each function of task1,2,3,4,5 worked normally through Test4.c. In particular, task1 was able to minimize internal fragmentation by efficiently managing unused memory space by applying bestfit algorithm.

4. Discussion

Through sshrink(), unused containers present at the end of the heap can efficiently manage memory by moving break points. However, Unused containers that exist between Busy containers will remain wasted on memory space. To solve this problem, add an API that performs storage operations at the right time to minimize wasted space. Through this method, smalloc can be implemented to efficiently manage memory space.

5. Conclusion

Through the process of implementing smalloc, we learned how to allocate memory by applying bestfit algorithm. They also learned how to compress sfree() as much free space as possible for efficient memory management. We also learned how to dynamically add or decrease memory through the process of implementing slalloc. In addition, I learned about memory copy operation using memcpy. Finally, through the process of implementing sshrink, we also learned how to determine the size of the heap by using sbrk() system call to adjust break point.