ITP 30002-02 Operating System, Spring 2020

Homework 5

Tae Eun Kim 21400217, 21400217@handong.edu

1. Introduction

The goal of this homework was to upgrade the given smalloc 1.0, a simple heap memory allocation library. Modifications were made in following features.

First, the allocating algorithm was modified from first-fit to best-fit. Second, at the free call, all unused containers that are adjacent to the target container that is being freed were merged. Third, srealloc() was implemented to reallocate the container of the given address with the given new size. Fourth, sshrink() was implemented to reduce the heap size as much as possible. Fifth, print_mem_uses() was implemented to print out the memory usage status.

2. Approach

Here is the simple overview of how this library virtualizes the Heap memory. Each memory block is virtualized as a container and Heap is virtualized as a linked list of these containers. Each container has a header which holds the metadata for that container, whether it is being used, address of the previous and next container, and the size of the data it can hold.

2.1 Allocation and free

In order to allocate new memory, we must seek for the best-fit hole by searching through the list of containers. If no containers are available, we retain more memory by asking the OS, make it into a new container, split it as we need, and allocate it by marking it Busy and returning its address.

To free memory, we simply find the target container and mark it as Unused. Then merge with adjacent containers if they are Unused too.

We might need more, or less, memory allocated for some pointer. Then we can either split the container and reduce the dsize in case of reduce, or try to expand the container. If the next container is unused and big enough, we can simply take it and split it, otherwise, we will need to retain more memory if it is busy or not big enough.

2.2 Others

This is how sshrink() works. Since Busy containers must not be removed nor even moved, what we can do is to let go of only the series of unused containers at the end of the list. To do so, I called sbrk() with the negative value corresponding to the sum of the size of unused containers at the end of the list.

For the print_mem_uses(), I went through the entire list of containers to retrieve information of current memory uses. I recorded the total size of all containers, including their header size, and sorted them according to their status. Then I printed out the total memory retained by this library, ones being used, and ones not being used.

3. Evaluation

Evaluation of this program was made in two approaches. First, are the new functionalities working correctly? Second, is the best-fit algorithm more memory-efficient than the first-fit algorithm? For the first approach, I checked many scenarios that covers different cases for each of the new functionalities.

For the second approach, I designed a test case, test4.c, to demonstrate a scenario where best-fit algorithm uses memory more efficient that the first-fit. After 4 smalloc() calls, best-fit algorithm resulted in 4 containers in 1 page, all busy, while first-fit algorithm had 6 containers across 2 page, yet 2 unused. Thus, smalloc 2.0 works correctly and is more efficient than the previous version.

4. Discussion

4.1 Possible Improvements

The biggest problem of smalloc 2.0 is that it only considers the page size, not the page alignment issue. There are two ways of improvement.

First, when retaining more memory with an empty list, we can consider the current entire memory usage, trying to align the heap with the actual page usage. Then we may retain memory in an actual page-wise manner. After on, we can retain memory by just considering the page size, since the heap is already aligned with actual page usage.

Second, when shrinking the heap size, we may shrink it while regarding the page size (page alignment). Since trivial shrinks or ignorant shrinks may result in overhead or fragmentation, it would be more efficient to shrink the heap according to the page size. If not, we have to align the heap with the page size again.

5. Conclusion

I built a program that extends the smalloc 1.0. By virtualizing the heap memory into containers, I could manage them efficiently in a similar manner to the way OS treats the virtual memory.