ITP 30002-02 Operating System, Spring 2020

Homework 5

21800500 Goeun Lee, 21800500@handong.edu

1. Introduction

This task is to create functions that retain, allocate, and deallocated memory in the heap, such as the malloc and free of <stdlib.h> library. The structure _sm_container_t is a unit of a series of operations, such as allocating and deallocating memory for a heap. The member variables of the structure _sm_container_t include a variable repre-sentting the status consisting of an enum of 'Busy' and 'Unused', a pointer variable pointing to the previous and next nodes, and a size representing the memory size of the current node. Each node forms a double linked list. The functions smalloc(), sfree(), and print_sm_containers were already implemented. The management of memory that my program has been allocated from the heap is up to my program, if there is an area that I can give from the memory I manage when the user requests smalloc, I can listen to the request without allocating new heap memory. Retained memory means the memory that my program has been allocated from the heap, that is, the memory area that my program manages. Allocated memory refers to the memory area assigned to the user within the control area of my program. All memory areas are calculated by the dsize of each node, and it should also be considered that each node has as many as 32 bits of headers. A node is defined as a structure _sm_container_t structure and its size is 32bit.

2. Approach

The first task is to change the way smalloc() is implemented from first-fit to best-fit. In the process of checking all nodes one by one, if the status is Busy, just pass the node, but if not already assigned, store it in the hole immediately. To implement as best-fit, we should not set the hole as the first node and stop the check, but instead compare the size of the node and the size of the node that we want to be assigned by going around the foreword and finally store the smallest difference in the pointer called hole. The second task is to merge two or more nodes that have become Unused after freeing. When asked free, not only changed status to Unused, but also connected double linked list if front and rear nodes are Unused. The third task was the implementation of print_mem_use, which obtained the dsize of all nodes, added the size of the header (sm_container_t) when calculating the retained memory, and the size of the node whose status is Busy was allocated memory and the node's dsize was unused by the node. The fourth task is the implementation of srealloc(). we have to change the size of a node that is already Busy. If the next node on that node is Unused and the sum of current node and next node is bigger than user's request, I implemented realloc by combining the next node with the current node and then split it to the desired size. If the next node is Busy or not large enough, free the current node and call the smalloc(). The 5th task is the implementation of sshrink(). I implemented the function as that if the last node's status is unused, return it to the heap as possible. If the last node is unused and the size of the last node is bigger than page size, I reduced the size of the memory that my program manages using brk().

3. Evaluation

When a user requests smalloc() in order of 2000, 2500, 1000, the node is created in order of busy2000/ unused-2032/busy2500/busy1000/unused500. If it were first fit, 1000 would have been assigned to the first Unused node, but because it was the best fit, it would have been assigned to the fourth node in the best size. If you sfree a 2,500-sized node and realloc a 1000-sized node to 1500, the node will be created in the order busy2000/unused-4564/busy1500. Because the last node is busy, there is no change even if you run sshrink(). Do sallocate() 1000, 5000 and free the node sized 5000. Then the node is created in order of busy2000 /busy1000/unused3532/ busy1500/unused8160. If you run sshrink() ,since the size of one page is 4096, the size of the last node would be 4064, subtracting 4096 from the original size 8160.

4. Discussion

This program is for one user. Threads in a process share a heap. So, If the computer runs multi-threaded program, the operating system has to mange the heap for multi users. I think this program can be modified for multi-users. When I delete one node from a linked list in general c-coding, I free that node, but I wonder how to handle it in this task. I tried to use free() in <stdlib.h> library, but It made some errors.

5. Conclusion

For this assignment, I should have understood how heap allocates and deallocates memory for a program, and how the program allocates and deallocates memory for a user. heap allocate memory by moving pointer that points the end of the heap in page size units. The program calculated whether the memory is allocated in free units.