Operating System PA4 REPORT

21700646 Jeon Hye Won

1. Intro

A. Problem Analysis & Solution Overview

a. Version 1.1

i. Task 1-1

- In order to implement the function `print_sm_use()`, it was necessary to determine what each (1), (2), and (3) meant in assignment description. First, (1) means the total memory space retained by the `smalloc`. In other word, this is the total memory space that `smalloc` requested through the `sbrk` function to allocate space to OS. And (2) is the space allocated by `smalloc` to the request of the user, which is a busy space. Finally, (3) is the space currently retained by Smalloc, but has not yet been allocated to the user which is unused space.

ii. Task 1-2

- The code of the version 1.1 is implemented with first-fit algorithm, which allocates memory to unused and fittable space. However, Modification code to best-fit algorithm should traverse all container in order to find best container, and it maybe takes more time than first-fit algorithm. (I will modify code in Version 1.2 to traverse with more efficient) I simply traverse all container, and found out the best case. And allocations result in the same size space. In such a case, space was allocated to the container that came out first so that it could be stacked as neatly as possible.

iii. Task 1-3

- To show that best-fit algorithm is more efficient than first-fit algorithm, I allocate memory and then show that there is less internal fraction when space is allocated according to the best-fit algorithm than when space is allocated to the first-fit algorithm.

b. Version 1.2

i. Task 2-1

- As mentioned earlier, in the case of best-fit algorithms, pointer must traverse through all the containers in order to find the optimal hole. However, there are some of the containers that have already been allocated space, so there is no reason to traverse. This task will add additional next_unused fields to the existing sm_container_t structure so that I can only traverse the possible containers I can actually put in.

ii. Task 2-2

- In fact, if you repeat allocations and free, there is a lot of fragments in between the memories. However, leaving these spaces may be make bad situations. For example, if it is marked `Unused` and has three consecutive containers, it will not be able to allocate 3,000 bytes of memory even though it is actually available. (If you add the size of the container, you can actually allocate more than 3000 bytes.) To address these limitations, we will revise the `sfree()`, adding a field called prev to the `sm_container_t` for the efficiency of time.

2. Approach - Solution Design

A. Version 1.1

a. Implementing print_sm_uses( )

- To determine the amount of memory currently allocated, it was necessary to traverse the whole container chain from `sm_first`. At this time, I checked the status of each container to see if it is `Busy` or `Unused`. (to make sure that it was actually retained or not yet allocated) the total usage of memory was counted for each status, and finally, the total number of containers was also determined to output the total amount of memory.
- Basically, the total sum of (2) and (3) should be (1), but because of the size of the container, (1) is slightly larger.

b. The reason best-fit algorithm is better than first-fit algorithm

- The better performance of either algorithm can actually be interpreted in two ways. One is that it takes less time to perform an algorithm, and the other is the portion associated with memory usage. However, the `test4.c` example illustrates when the best-fit algorithm is faster than the first-fit algorithm in the latter case. `test4.c` allocates 2000, 2500, 750 and 750 in turn. In best-fit, the remaining internal fraction after allocation is combined into 2032, but in first-fit, it is divided into two sections: 468 and 1532 respectively. 2032 bytes may be more useful than 468 byte and 1532 byte when other requests are received and additional allocations are required.

B. Version 1.2

a. Add new field to `sm_container_t`

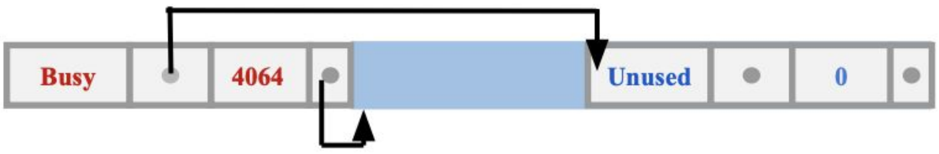
- It is inevitable that the best-fit algorithm should traverse the entire container given. Instead, I tried to reduce the number of traversal containers in order to traverse in the most effective way. I use the new field `next_unused` here. This field has the location of the container that can allocate space, not just the next container, but the one that is unused, among the contacts behind me.
- Performing Task 2-1 and Task 2-2 required more pointer operations than Version 1.1. So, I judged that it was inefficient to check whether the status of the status and the condition of the all container. For this reason, in Version 1.2, I added another field called `prev` not only `next_unused`. Using this field, the connection between the containers could be implemented as a doubly linked list rather than as a single linked list, and the calculation time could be reduced further.

b. Maximize unused memory space

- If more than one unused space is existed, it should not be allocated to that space and should be allocated to another space, even though it may be allocated a larger memory space. This is inefficient. So whenever the memory frees, it uses field `prev` and `next` to check whether their status is unused and merge them, allowing them to use the memory more efficiently.

c. Improving `smalloc`

i. There's a case we didn't want. When the size of the container is added, the remaining space becomes zero and becomes meaningless. In this case, I thought it would be better to automatically request additional pages so that no meaningless container exists.



ii. Now, I allocates space from linear logical addresses when allocating space. Of course, this may be separated from the physical address, but the problem is that since it is continuous in the logical address, efficient memory management is difficult. Of course, additional computations will be required to merge unused container, but it will be more efficient if the container is segmented and stored without continuous space.

제 Lab-2019E-149846 호

수 료 증

소　　속 : 한동대학교
성　　명 : 전혜원
이수과정 : 2019 전기 안전교육
이수기간 : 2019년 01월 01일 ~ 2019년 06월 30일 (2시간)

위 사람은 과학기술정보통신부에서 주최하고 한국생명공학연구원 국가연구안전관리본부에서 주관한 2019 전기 안전교육을 수료하였으므로 이 증서를 수여합니다.

2019년 06월 15일

한국생명공학연구원 국가연구안전관리본부장

한국생명공학연구원국가연구안전관리본부장인