HO CHI MINH NATIONAL UNIVERSITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS

Report

Lab 04 Process

Student: Duc Nguyen Quang 1810118

HO CHI MINH CITY, 5/2020

Contents

1 Explain the exercise 1

1.1 Align malloc

At the time align_malloc is called, it will check if the head of the heap it manages is NULL or not. If the head is NULL, align_alloc will call request_space. Otherwise, it will find first fit free block. If there is no free block, it will call request_space and otherwise return that free block.

1.2 Request space

If this function is called, it means we need a new area for requested space. Therefore, we get the break 's current address, then we find the nearest address that divide align varible. Then we crate a block for storing metadata right before the found address. After that, we link the previous block to this block and return this block.

1.3 Find a free block

Because we need to find a free block that contains an address divide align varible and have the size from that address to the end of the block larger then the requested space, we check out all the free block. When we found the first block that meets the above conditions, we crop the space from the block start to the address and join it with previous block. Then we modify the current block's metadata and return it. If there is no satisfied block, we return NULL.

1.4 Free a pointer

When we free a pointer, we first get it's metadata lie right before it. Then we set it's free attribute to true. For the most efficient memory use, we merge all free block the lie next to each other into 1 big block.

1.5 Merge free blocks

At first, we find a free block and check if it successor blocks are free. We will count the new size of the big block by sum all size of this and successor blocks. And then we re-link the first free block to the first unfree block and re-set it's size.

2 Exercise 2

At first, we must know that the builtin malloc and free method of standard library in C use 2 seperated linked list for allocated address and free address. Compare to my implementation, the idea of memory management is also the same. Both have a function to find free space, a function to move the break, a function to merge free blocks. But with builtin malloc and free, the requested memory can be allocated more quickly and more suitable due to using 2 linked lists and best-fit method. While my implementation only uses 1 linked list, so that, it can be developed easier but it will not be as efficient as the builtin due to first-fit method.

Secondly, this implementation uses an additional condition that is the return address must divide the pre-defined align varible. Therefore, it can create some more gap between allocated spaces. It is surely bad for efficient memory using but it's safer for different spaces isolation and stable operating if being correctly used.

Due to all advantages and disadvantages of both builtin and my malloc implementation listed above, we can consider that the builtin malloc can be very suitable for more dynamic data. That means it will allocate and deallocate many many times and due to its model, the memory can be managed more optimally. While my implementation with align method can make processes more stable, reduce memory leak when using with less dynamic data (less allocating and deallocating on 1 pointer).

When a process needs more heap space because it has run out of allocated space, the standard library will issue a call to the OS. The OS will allocate a page and it will be put in free list, which usually will be manage by user library for the program to use. For example, if the program wants 1 KB more, the OS will give an additional 4 KB (a page) and the library will give 1 KB to the program and have 3 KB left for further use.