Grenoble Universities (GrenobleINP & Université Joseph Fourier)
M1 MoSIG
Academic year: 2014-2015

Operating System Design — Midterm exam – ANSWERS

October 22, 2014 — Duration : 90 minutes

Note : For pedagogical purposes, most answers are more detailed than what was expected for the exam.

Problem 2

Question 2.1

First, as seen in the lectures, we can notice that it is necessary to store the block size in the header, both for free blocks and allocated blocks. For free blocks, this is obviously required to determine if a block is big enough to satisfy a `malloc` request. For allocated blocks, this is necessary to discover the size of a block that must be freed, because there is no such information passed as input to the `free` function.

Now, let us discuss the crux of the question, i.e., the motivation for (i) the `a/f` (allocated/free) boolean in the header and (ii) the footer that duplicates the contents of the header. Let us consider three adjacent blocks in memory : B_1, B_2 and B_3 and let us assume that B_2 is allocated. When B_2 is freed by the application, the allocator must check the state of B_1 and B_3, to determine if they can be coalesced with B_2 in order to form a larger free block. The chosen block format allows implementing these checks in a efficient way.

— To find out about the state of B_1, the allocator simply needs to read B_1's footer, which can be found on the 4 bytes that are immediately before the start of B_2. And if B_1 is indeed free, performing the coalescing of B_1 and B_2 requires determining the starting addresses of B_1, which can be easily computed using B_1's size (found in B_1's footer) and the address of B_1's footer.

— Similarly, to find out about the state of B_3, the allocator simply needs to read B_3's header, which can be found on the 4 bytes that are immediately after the end of B_2.

Question 2.2

Minimum possible size for an allocated block First, notice that an allocated block must necessarily have a payload size greater than zero (otherwise, the block would be useless). To comply with alignment constraints, we need to make sure that, within an allocated block, the first address of the payload is always a multiple of 8. Let us consider the following example : two adjacent allocated blocks B_1 and B_2 at the start of the heap. Let us call P_1 (respectively P_2) the first address of B_1's payload (resp. B_2's payload). According to the specifications, we know that P_1 is a multiple of 8. Now, let us determine the minimum possible size S for B_1's payload + padding. To enforce the alignment constraint for P_2, the size of the gap between P_1 and P_2 must be a multiple of 8 bytes. This means that S + *footer size of* B_1 + *header size of* B_2 = $8.N$. Which gives us $S + 4 + 4 = 8.N$. Given that S cannot be null (as explained previously), the smallest possible value for S is 8. Therefore, the minimum size for an allocated block is 16 bytes (size of header + S + size of footer).

Minimum possible size for a free block Given that there is no payload and that all the individual fields (header, pred pointer, next pointer, footer) have the same size (4 bytes), there is no specific

additional alignment constraint for a free block [1]. Thus, no padding is necessary. And the minimum size is the sum of the sizes for the different fields : 16 bytes (of which only 8 can be used by the application).

Additional remarks

— In general, the minimum size for an allocated block should not be smaller than the minimum size for a free block (and vice versa), because an allocated block can later be freed (and vice versa).

— Note that, due to alignment constraints, allocated blocks (and also free blocks, because an allocated block may later be freed) cannot have arbitrary sizes equal to or greater than the above-discussed lower limit (16 bytes). For example, let us assume an initially empty heap and a first `malloc` request for 19 bytes. Allocating only a total of 27 bytes (4-byte header + 19-byte payload + 4-byte footer) is not possible because the payload of the next block would not be properly aligned. Hence, a padding area of 5 bytes must be added after the payload area, in order to form a 32-byte block. More generally, the size of the padding must always be chosen so that the starting address of the next block is a multiple of 4 and not a multiple of 8 (i.e., in the form of 8.X + 4. For example : 4, 12, 20, 28, etc.). In this way, the starting address for the payload of the next block will be a multiple of 8 (see footnote for details [2]), which is necessary because this next block may become allocated. Overall, this boils down to the following rule : the whole size of an allocated block must always be chosen so that it is a multiple of 8.

Question 2.3

2.3.1

The following sequence of malloc requests will allocate all the heap heap space in the quickest way (least number of requests), all while requesting the least amount of space :

— malloc(1) → 4 kB from 1st half of heap
— malloc(1) → 4 kB from 1st half of heap
— malloc(1) → 4 kB from 1st half of heap
— malloc(1) → 4 kB from 1st half of heap (now, the 1st half is fully allocated)
— malloc(16384) → allocate the whole 2nd half in one shot

2.3.2

Possible types of fragmentation :

— external fragmentation : for the second half of the heap (because we may have blocks with different sizes and different lifetimes) ;

— internal fragmentation : for the first half of the heap (because we systematically round up small requests to 4 kB) and possibly also in the second half of the heap (if we use padding, e.g., due to alignment constraints).

1. We assume that the first address of a free block is always a multiple of 4, which stems from the way allocated blocks are reserved. The whole size of an allocated block must always be chosen so that the first address that is just after the allocated block is a multiple of 4 (and not a multiple 8). See the "additional remarks" section for details.

2. Indeed, given that the starting address of the block is 8.X + 4 and that the header size is 4 bytes, the starting address of the payload will be 8.X + 4 + 4 = 8.(X+1), which is a multiple of 8.