Name: Olivér Izsák

AIS ID: 112294

Slovenská Technická Unvierzita v Bratislave Fakulta Informatiky a Informačných Technológií

Data Structures and Algorithms

Assignment 1. – Memory Manager

by Olivér Izsák

AIS ID: 112294

Academic year: 2020/2021

9.March 2021

Name: Olivér Izsák

AIS ID: 112294

1.Memory allocation

I implemented an implicit solution to the malloc assignment, which means the blocks of memory are stored using an array. More precisely, I used circular linked list within the memory to keep track of the pointers. My malloc solution was implemented through 6 methods. 4 of those were already given : memory_init, memory_alloc, memory_free, memory_check and I added a 5th method called "bestFitStrategy" and a 6th method called "coalesce".

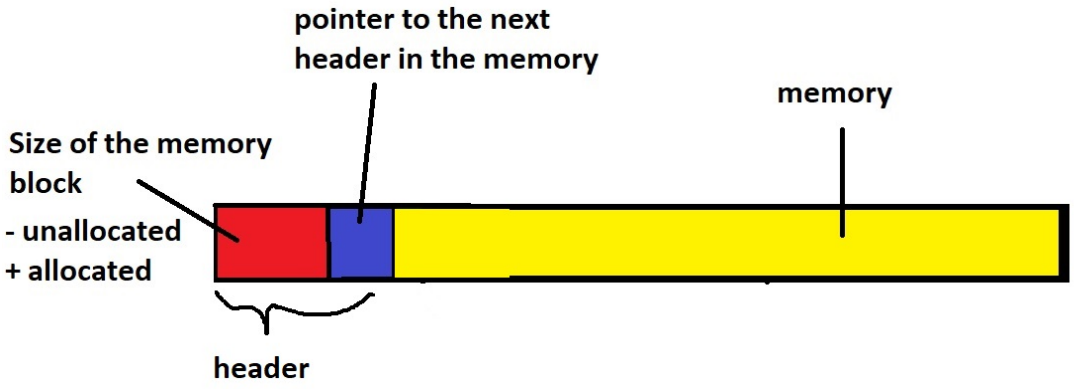
I used structures called "headers" where I stored the pointers of the next header and an int size, which could be positive(allocated memory) or negative(unallocated memory) and stored the size of that memory block. :

```
struct header
{
        int size;
        header* next;
};
```

My global variable: `void* mem;`

1.Method : void memory_init (void* ptr, unsigned int size)

Here I assign the ptr pointer to the memory. Then I create the main header in the memory. Give it the size of the global memory minus the size of the header which is 8 bytes. Lastly, I assign the next pointer itself, because I am using a circular singly linked list.



2.Method : void bestFitStrategy(int size)

Firstly I create a header pointer and give it the value of the main header in the memory. I create a 2 temporary variables that will hold the values of different memory blocks and one for their size.

After there is a while loop, which goes through my linked list and tries to finds a free block of memory that can contain the memory I want to allocate. It will always find the memory

block that is closest in size range to the one being allocated, hence the name best fit strategy. If free memory block big enough in size is not found then it returns NULL.

```
while (checkUniqueCase==0) { // 1111111111111111111
    if ((*localMem)->next == (header*)mem) {  checkUniqueCase = 1; }// 1111111
    if (-((*localMem)->size) >= size) {
        if (-((*localMem)->size) == size) { bestFit = (*localMem); break; }
        if (i == 0)
        {   //we take the first header that can contain the memory, and we keep
            tempSize= -((*localMem)->size);
            bestFit = (*localMem);
        }
        if (-((*localMem)->size) < tempSize )
        {
            bestFit = (*localMem);
            tempSize = -((*localMem)->size);
        }

        i++;
    }

    localMem = &(*localMem)->next;    // we go to the next block of memory
}
```

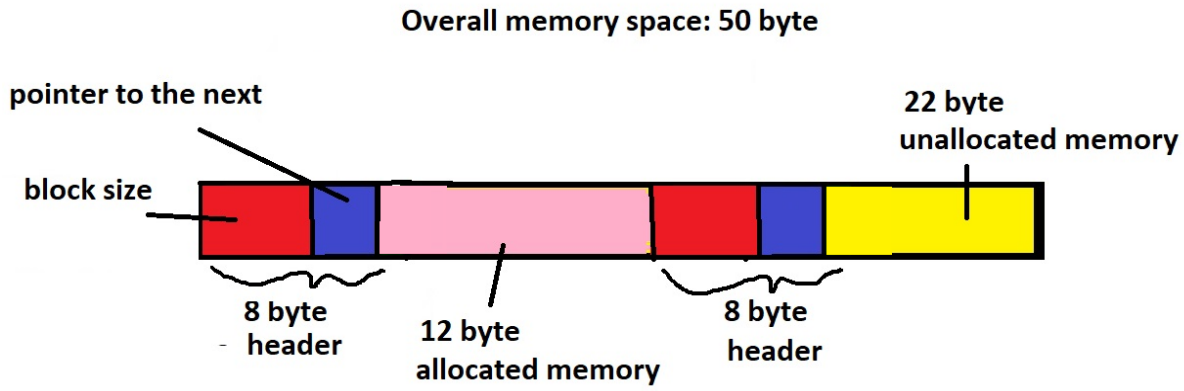
3.Method : void* memory_alloc (unsigned int size)

In this method I use my other method called “bestFitStrategy”.
In memory_alloc method I have one method variable called size. Which is a positive integer that my method has to allocate.
Through the “bestFitStrategy” I find the most suitable memory block where I can allocate this size of memory and return it’s pointer to the header. Then I calculate the amount of space that is left after allocating eg.: If the best block the bestFitStrategy has found is 42, and I want to allocate only 12, then here I will substract that 12 from 42, which means I still have 30 unallocated memory left in that block. In this case I will create an entirely header for (8 byte) and to that a corresponding new unallocated memory block containing 22 bytes of free space. If the memory left after the initial allocation is smaller than 8 bytes (the size of my header) then that memory becomes fragmented and just allocate it together with the rest. This happens because if I have 8 or fewer bytes of memory left, I cannot create a new block of memory.

Name: Olivér Izsák

AIS ID: 112294

Visualization:



4.Method : void memory_free (void* valid_ptr)

In memory free I get a ptr parameter and first check whether that pointer can be found in my memory through the memory_check method. If the memory_check returns 1, which means it's a valid pointer then I get the pointer to the end of the header for that pointer and call the coalesce method. After that returns the size of the memory that has been allocated. I will elaborate on the memory_check and coalesce method below.

5.Method : void memory_check (void* ptr)

First I create a pointer to the header struct and give it the value of the main header. Then I create a while loop where I go through the list of all my headers and try to find one that corresponds to the one given by the ptr parameter. I also check if the size not zero, because if it is there is no reason free a memory that's already freed.

```
int memory_check(void* ptr)
{
    header** localMem = (header**)&mem;
    if (ptr == NULL) {  return 0; }

    int checkUniqueCase = 0; // this is for the situation when a header actually points to
    while ( checkUniqueCase == 0) {
        if ((*localMem)->next == (header*)mem) { checkUniqueCase = 1; }

        if (((char*)ptr - sizeof(header)) == (char*)(*localMem) && (*localMem)->size > 0)/
        {
            printf("Memory authentication...\n");
            printf("Valid memory address\n");
            return 1;
        }
        localMem = &(*localMem)->next;
    }


    printf("Invalid memory address\n ");

    return 0;
}
```

Name: Olivér Izsák

AIS ID: 112294

6.Coalesce: int coalesce(header** memFree)

Here I basically get the size of the memory block I am about to unallocated and negate it, so It is already shown as unallocated memory. Then I check whether the pointer to the next is either the same as the main header or whether the actual memory block is the same as the main header. If they are then I am not going to join the previous or the next one to them, because then it would corrupt the main header to the whole memory I have. If they are not the same as the main header then I first go and check the next block of memory whether it is unallocated or not. If yes, then I join the two.
After that I also check the previous block of memory.
Using the circular linked list I created I go through the list using a loop until I end up at the memory block which next points to the one I started from. After that I check whether that memory is unallocated (negative) if yes, then I join them together by adding the next ones size to the current one and adding the next one's pointer to next to the current one's next. This sounds complicated in written format so here is the code and below that a visualization of the join process:

Joining the next to the current one:

```
int coalesce(header** memFree) { //  defragmentation
    int size, nextCounter = 0;

    size = -((*memFree)->size);

    printf("Memory defragmantation in process...  \n");
    printf("Checking the surrounding blocks of memory for coalescing...\n");
    if ((*memFree)->next != (header*)mem && (*memFree) != (header*)mem) {

    if ((*memFree)->next->size < 0) {// if the next block of memory is free

        printf("Found free memory in next block. Size of memory: %d\n", -((*memFree)->next->size) + sizeof(header));

        size = (size + ((*memFree)->next->size - sizeof(header)));
        (*memFree)->next = (*memFree)->next->next;

        printf("Overall freed memory size : %d \n", -size);

    }
```

Joining the current one to the previous:

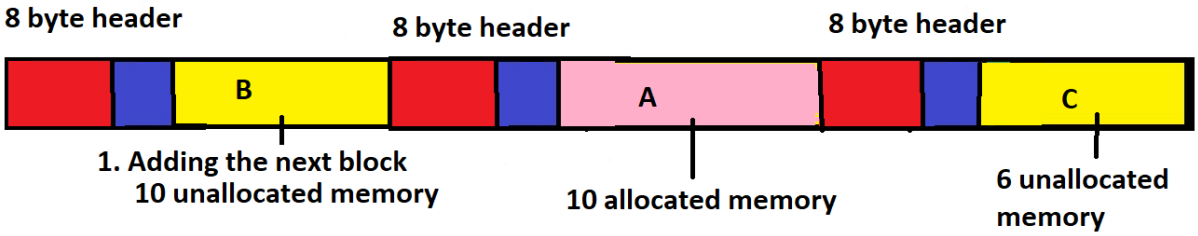
```
    header* tempPointer = (*memFree)->next;

    int sizeOFThePrev = 0;

    while (tempPointer->next != (*memFree))
    {
        tempPointer = tempPointer->next;

        sizeOFThePrev = (tempPointer)->size;
    }

    if (sizeOFThePrev < 0 )
    {

        printf("Found free memory in previous block. Size of memory: %d\n", -(sizeOFThePrev) + sizeof(header));
        size = (size + (sizeOFThePrev - sizeof(header)) );

        tempPointer->next = (*memFree)->next;
        tempPointer->size = size;
        printf("Overall freed memory size : %d \n", -size);
    }
    }
    else    printf("Overall freed memory size : %d \n", -size);

    return (size); // here we convert it to negative , since negative numbers mean unallocated memory.
```

Name: Olivér Izsák

AIS ID: 112294

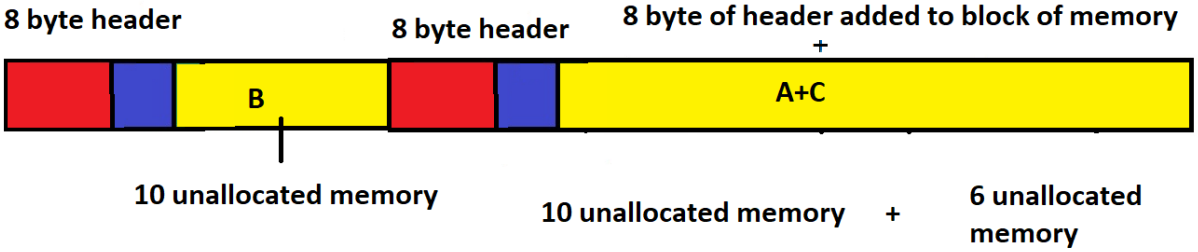
Visualization:

Overall 50 bytes of memory space

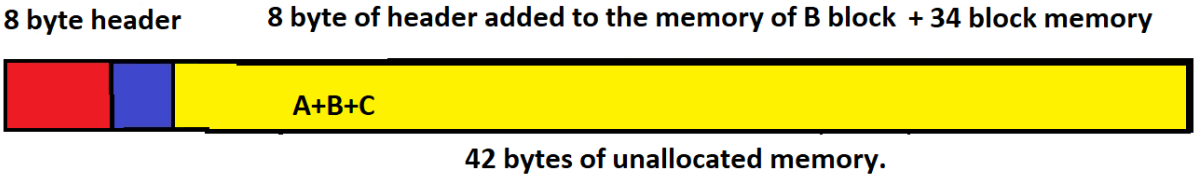


Freeing memory block A adds the size of the C memory and header to memory block A

1. **Adding the next block to it**



2. **then we are adding it to the A block**



2.Comments on my assignment:

Circular linked list:

In my opinion using a circular linked list was a more space efficient way of implementation since in the header structure this way only costs 8 bytes, while doing it with doubly linked list would increase it to 12 bytes.

Best fit strategy:

If it comes to space efficiency then this strategy is the best. I could have also used first fit. Which would increase time efficiency.

3.Time and Space complexity estimate:

Memory allocation is possible to be done in constant time complexity, O(1).

I didn't manage to make it in constant time, since I am using loops to go through my circular linked list. This means my time complexity is linear, O(n)

My space complexity is also linear since the circular linked list is not constant, but increasing, O(n).

4.Testing:

I did 3 different kind of test.

Test one:

First one is a specific test, where I test the most important functionalities, such as allocating, deallocating memory, joining previous and next memory blocks.

Results of the test:

Name: Olivér Izsák

AIS ID: 112294

```
Memory initialization ...
 Memory initialized.
 Free memory: 92
 Header size: 8



Memory allocation starting...
Trying to allocate 15 byte of memory
Starting best fit algorithm...
The algorithm found a free memory with the size of: 92
Memory allocated
Fragmentation occured
Remaining unallocated memory within the block : 77 byte
Creating new block of memory from the remaining 77 byte of space
New header created:
 size : 8 byte
 size of new unallocated memory block: 69  byte



Memory allocation starting...
Trying to allocate 10 byte of memory
Starting best fit algorithm...
The algorithm found a free memory with the size of: 69
Memory allocated
Fragmentation occured
Remaining unallocated memory within the block : 59 byte
Creating new block of memory from the remaining 59 byte of space
New header created:
 size : 8 byte
 size of new unallocated memory block: 51  byte



Memory allocation starting...
Trying to allocate 1 byte of memory
Starting best fit algorithm...
The algorithm found a free memory with the size of: 51
Memory allocated
Fragmentation occured
Remaining unallocated memory within the block : 50 byte
Creating new block of memory from the remaining 50 byte of space
New header created:
 size : 8 byte
 size of new unallocated memory block: 42  byte
```

Name: Olivér Izsák

AIS ID: 112294

```
Memory allocation starting...
Trying to allocate 22 byte of memory
Starting best fit algorithm...
The algorithm found a free memory with the size of: 42
Memory allocated
Fragmentation occured
Remaining unallocated memory within the block : 20 byte
Creating new block of memory from the remaining 20 byte of space
New header created:
 size : 8 byte
 size of new unallocated memory block: 12  byte



Memory allocation starting...
Trying to allocate 300 byte of memory
Starting best fit algorithm...
Memory is full.

Test for memory allocation : allocated: 48,headerspace: 32  =  80




Memory allocation starting...
Trying to allocate 8 byte of memory
Starting best fit algorithm...
The algorithm found a free memory with the size of: 12
The memory has been fragmented.
 There is 4 byte of space left that cannot be allocated
```

Name: Olivér Izsák

AIS ID: 112294

```
Memory authentication...
Valid memory address
Memory defragmantation in process...
Checking the surrounding blocks of memory for coalescing...
Memory freeing process done.
Amount of memory freed: 1
Starting memory free process
Memory authentication...
Valid memory address
Memory defragmantation in process...
Checking the surrounding blocks of memory for coalescing...
Found free memory in previous block. Size of memory: 9
Overall freed memory size : 31
Memory freeing process done.
Amount of memory freed: 31
Starting memory free process
Memory authentication...
Valid memory address
Memory defragmantation in process...
Checking the surrounding blocks of memory for coalescing...
Found free memory in next block. Size of memory: 39
Overall freed memory size : 49
Memory freeing process done.
Amount of memory freed: 49



Memory allocation starting...
Trying to allocate 41 byte of memory
Starting best fit algorithm...
The algorithm found a free memory with the size of: 49
The memory has been fragmented.
 There is 8 byte of space left that cannot be allocated



Memory allocation starting...
Trying to allocate 6 byte of memory
Starting best fit algorithm...
Memory is full.



Allocated blocks of memory:  64
allocated headers : 24
All memory used: 88
Allocated blocks percentage compared to the ideal solution: 88 / 100 = 0.88
Percentage of actual memory space allocated(excluding headers): 100 / 64 = 0.64
Percentage of fragmented memory space 12 / 100 = 0.12
Percentage of header memory space 24 / 100 = 0.24
```

Name: Olivér Izsák

AIS ID: 112294

Test two :

Second one is testing the allocation with random numbers from a scale, which can be given by the user (8-24,500-5000 etc…). The size of the overall memory can also be given. eg.:50/100/200/50000 etc…

This is how the test looks:

```c
int main() //test with random values no frees.
{
    time_t t;
    int allocatedMem = 0;
    int headerMem = 0;
    int const min = 500;   // here change minimum to 500 and max to 5000 or 8-50000
    int const max = 5000;
    int const regionSize = 100000;  // here change regionsize to 50/100/200 or 50000+
    int const n = (regionSize / (sizeof(header) + min)) + 1;  // amount of times to run the for loop

    char region[regionSize];
    memory_init(region, regionSize);
    char* pointer[n];
    int count = 0;
    srand((unsigned)time(&t));
    for (int i = 0; i < n; i++) {
        int mem = min + rand() % (max - min) + 1;


        pointer[i] = (char*)memory_alloc(mem);

        if (pointer[i] != NULL) {
            memset(pointer[i], 0, mem);
            headerMem += 8;
            allocatedMem += mem;

            count++;
        }
```

This is the result:

```
Allocated blocks of memory:  99693
allocated headers : 280
All memory used: 99973
Allocated blocks percentage compared to the ideal solution: 99973 / 100000 = 0.999730
Percentage of actual memory space allocated(excluding headers): 99693 / 100000 = 0.996930
Percentage of fragmented memory space 27 / 100000 = 0.000270
Percentage of header memory space 280 / 100000 = 0.002800
```

Name: Olivér Izsák

AIS ID: 112294

Test three:

The last one where I am testing the allocating then freeing and allocating again with different big/small and random numbers and comparing it to the ideal solution.

Case 1: -Max 50 memory size
-Allocation random 8-24

```
Allocated blocks of memory:  28
allocated headers : 16
All memory used: 44
Allocated blocks percentage compared to the ideal solution: 44 / 50 = 0.88
Percentage of actual memory space allocated(excluding headers): 28 / 50 = 0.56
Percentage of fragmented memory space 6 / 50 = 0.12
Percentage of header memory space 16 / 50 = 0.32
```

Case 2: -Max 100 memory size
-Allocation random 8-24

```
Allocated blocks of memory:  48
allocated headers : 48
All memory used: 96
Allocated blocks percentage compared to the ideal solution: 96 / 100 = 0.96
Percentage of actual memory space allocated(excluding headers): 48 / 100 = 0.48
Percentage of fragmented memory space 4 / 100 = 0.04
Percentage of header memory space 48 / 100 = 0.48
```

Case 3: -Max 200 memory size
-Allocation random 8-24

```
Allocated blocks of memory:  98
allocated headers : 72
All memory used: 170
Allocated blocks percentage compared to the ideal solution: 170 / 200 = 0.85
Percentage of actual memory space allocated(excluding headers): 98 / 200 = 0.49
Percentage of fragmented memory space 30 / 200 = 0.15
Percentage of header memory space 72 / 200 = 0.36
```

Case 4: -Max 50000 memory size
-Allocation random 500-5000

```
Allocated blocks of memory:  45443
allocated headers : 240
All memory used: 45683
Allocated blocks percentage compared to the ideal solution: 45683 / 50000 = 0.91
Percentage of actual memory space allocated(excluding headers): 45443 / 50000 = 0.91
Percentage of fragmented memory space 4317 / 50000 = 0.09
Percentage of header memory space 240 / 50000 = 0.00
```

Case 5: -Max 50000 memory size
-Allocation random 8-5000

```
Allocated blocks of memory:  42179
allocated headers : 320
All memory used: 42499
Allocated blocks percentage compared to the ideal solution: 42499 / 50000 = 0.85
Percentage of actual memory space allocated(excluding headers): 42179 / 50000 = 0.84
Percentage of fragmented memory space 7501 / 50000 = 0.15
Percentage of header memory space 320 / 50000 = 0.01
```

5.Conclusion

I think my program is not the most efficient when it comes to time complexity, because it could have been done with constant time complexity.
It could have been done better if I gave the headers more information so that I wouldn't have to go through the circular singly linked list every time I wanted to allocate or free.
I think my tests are good enough. I covered the most important parts of my algorithm, which initializing, allocating, freeing, checking memory, and also checked whether the coalescing of surrounding free blocks of memories.
I also tested small and large memory allocations and their efficiency to the ideal solution are not far away.
In conclusion, my program could have been done explicitly which would have been a lot more efficient both space and time wise and even when it comes to my implicit solution it could have been more time efficient by making it constant in time complexity.