https://folk.ntnu.no/michaeng/tdt4186_21/
michael.engel@ntnu.no

Practical exercises
Spring 2021

Practical Exercises 3

Processes

Please submit solutions on Blackboard by Thursday, 04.03.2021 12:00h

The file `mymalloc.c` contains a C program skeleton for a heap allocator. To keep things a bit more simple, the heap is implemented here as a simple array of bytes with 64 kB.
In this exercise, you will implement your own *heap memory management* functions `void *mymalloc(int size)` to allocate a block of memory of the given size from the heap and `void myfree(void *p)` to release the block of memory pointed to by the pointer parameter, e.g.:

```
void *p;
p = mymalloc(42);         // allocate 42 bytes
if (p != (void *)0) {
  // do something
  myfree(p);              // release memory again
} else {
  printf("mymalloc failed!\n");
}
```

a. Implement the `mymalloc` function. (4 points)

   For the implementation, use the *first fit* algorithm discussed in lecture 9 on slides 12 and 13. Here, the linked list of free blocks is stored in the free memory blocks themselves.

   In order to guarantee that there is sufficient memory available for storing the metadata, each allocation of a memory block allocates the requested memory size (ensure that you round up the allocated size to a multiple of 8 to ensure correct memory alignment) *plus* the size required for the metadata, which is contained in a `struct mem_control_block`. Initially, there is only one free block which covers the complete memory space.

   To find a free block, iterate through the list of free blocks. A pointer to the first element in the list If the first found block is larger than the allocation, split the block into two pieces (the newly allocated block and the remaining free space) and link the remaining space to the free list.

   If no block can be found to fulfill the allocation, return `(void *)0`.

   Hint: This exercise requires you to work intensely with *pointer arithmetics*. Review the section in the C crash course if you are unsure about pointer behavior and arithmetic operations on pointers! Remember that adding a value to a pointer always adds `the size of the array pointed to` to the address. For the `void *` type, this offset is always 1 (i.e., you count bytes). You can *typecast* pointer types, e.g.

   ```
   struct mem_control_block *m;  // m+1 adds sizeof(struct mem_control_block) to m!
   void *p = (void *)m;          // p+1 adds 1 to p
   ```

b. Implement the `myfree` function. (4 points)

   When freeing a block, make sure to combine two adjacent free gaps in memory in order to form a larger block. Don't forget to add the freed block back to the free list.

c. Design and implement *test cases* for your `mymalloc`/`myfree` implementation. (2 points)

   The test cases should be implemented in the `main` function. Describe which situations can occur and what you do to check for the correct behavior of your memory allocator.

Note: It is sufficient that you submit a single source code file with the combined solutions for parts a, b and c.