Buddy Allocator within disastrOS

thanks to prof. Giorgio Grisetti

This project is about an implementation of Buddy Allocator with the auxiliary structure - i.e. bitmap - to store the available free blocks, added to disastrOS written by Giorgio Grisetti.

Buddy Allocator

It is implemented a Binary Buddy, so that the size of each block is a multiple of 2, and more precisely the size of each block of order n is proportional to 2^n, thereby each block are exactly twice the blocks that are one order lower. Each block therefore belongs to an order, which defines its size, there are 2^n blocks within the order n. The entire monolith block represents the order-0 and the others belong to the increasing orders.
Buddy Allocator implies to initially reserve a contiguous interval of memory, which is divided in smaller blocks to allocate thereafter. Thus, there must be a maximum size of memory allocatable (that's the size of interval), but also a minimum size (the lower limit), which should be small enough to minimize the average wasted space per allocation but large enough to avoid excessive overhead.
The main advantages of this allocator are the good avoidance of external fragmentation - because of best-fit policy - and that the allocation and deallocation are fast, especially the latter because the maximal number of compactions required are equal to log2(highest order) in the worst case. On the other hand, since the block sizes are fixed it suffers from internal fragmentation.

Overhead

The total overhead used is equal to the size occupied by the bitmap plus 8 bytes (size of 2 integers) for each block allocated. These 8 bytes represent the preamble in which it is stored the size of the block - useful for the free function upon deallocating - and the index of the corresponding bit within the bitmap buffer.

Bitmap

The structure used to store free and allocated blocks is made up of a C struct which comprises the pointer to the buffer (array of bits) - whereby each entry is related to a block: 1 means that block is marked as used and 0 as freed instead - and two integers, one to store the actual number of bits that are in the buffer and the other one to store the size in bytes of the buffer. Get and set methods for the bits are implemented through some bitwise operators.
Bitmap makes the research of particular entries easier, such as the buddy (index + 1), parent (index / 2) and children (index * 2 and index * 2 + 1) by taking advantage of some properties of the binary tree related to the blocks of memory managed.

Implementation

Initialization At the beginning default parameters are set (configurable in "balloc.h" file), bitmap buffer of the proper size is allocated dynamically in heap, whereas the bitmap struct is put in .bss (known at compile time), and finally it is reserved a contiguous block of memory in heap for future allocation.
Allocation During memory allocation (balloc function) it is found out the correct level based on the bytes requested. After that, it finds the first free block (marked as 0) of that

level, if its parent is already SPLIT (marked as 1) means that the other buddy is used, so simply get the block found, otherwise (parent marked as 0) SPLIT the parent by marking it as used (bit equal to 1) and all his parents above it, and then get the block found. In both cases set as used recursively every child until the end of each subtree.
Just before returning the pointer to the allocated region, it adds the preamble of 8 bytes to the actual start of block and then provide the pointer moved forward by 8 bytes offset.
Resize In case there is not enough available memory during allocation, a Buddy resize comes into play, a greater interval of contiguous memory will be used, if finds out how many times the memory actually used so far will be doubled (based on the bytes requested), a greater bitmap buffer of proper size (based on the memory that will be used) is allocated dynamically in heap, and finally the old one is deallocated. During this process the new parameters - such as memory used size and new number of levels (orders) - are calculated, the minimum block size stay invariant instead. Note: the Buddy resize allows to minimize the actual overhead represented by the bitmap buffer located in heap.
Free When bfree function is called the preamble is read - by moving backward the pointer provided by 8 bytes - and based on the size of block it is cleaned, then it is marked as unused. If even buddy is marked as freed they are COALESCED by marking as freed also the parent block, this check is made also for the parent of parent and so forth until a buddy used is found or the root is reached. Anyway also every child (subtree) is marked as freed. Note: during freeing each block is reinitialized to zero for protection reasons.
Static version A static version is implemented with a fixed block of memory allocated at compile time in .bss. In contrast to dynamic version, no Buddy resize is allowed, so programmer must know beforehand the bytes that will be used. This block size is configurable in “balloc.h” header, too. However, this version is a bit faster than the other one.

How-to-run

Just include the “balloc.h” header into your C file to use “balloc” and “bfree” functions as you would do with “malloc” and “free”. During compile you have to link (in addition to yours) the three object files “bit_map.o”, “block.o” and “balloc.o” generated by Makefile just by typing the “make” command from terminal, finally do not forget as last parameter “-lm” to link the needed math library, too.
Note: the allocator is developed and tested on Linux Ubuntu 16.04 LTS, it is not guaranteed that works as expected even in other versions or different operating systems.

written by **Gianmarco Fioretti.**