Project 3 - Buddy Memory Allocator

Overview

Kernel needs to allocate memory for user-level applications and the kernel itself. For user-level applications, it allocates a fixed size page frame at the time of each page fault. For kernel's own allocations---such as allocating DMA buffers---it may need to allocate multiple physically contiguous page frames. To serve these requirements, Linux kernel implements an allocator based on the buddy algorithm.

In this project, you will implement and evaluate a memory allocator based on the buddy algorithm.

Background

The buddy algorithm manages memory blocks of exponential sizes (e.g., 4KB, 8KB, 16KB, ...). For example, if 21KB is requested, then the buddy allocator will return 32KB of memory block (waste of 11KB). The allocator keeps a set of free-lists for each block size.

Below table illustrates the workings of Buddy Allocator. Consider we have 1024K(1MB) of memory and the smallest possible block in the system is 64K. We have four memory allocation requests A, B, C and, D

1. Init	1024K					
2. A=80K	A	128K		256K		512K
3. B=60K	A	B	64K	256K		512K
4. C=80K	A	B	64K	C	128K	512K
5. A ends	128K	B	64K	C	128K	512K
6. D=32K	128K	B	D	C	128K	512K
7. B ends	128K	64K	D	C	128K	512K
8. D ends	256K			C	128K	512K
9. C ends	1024K					

The allocation could happen in the following order.

1. Init - Initial state of the system, where we have a single block of maximum size.
2. A allocates 80K of memory.

   a. The smallest possible block that can accommodate the request is 128K.
   b. No block of size 128K is available. If the free list of this block is empty, the buddy allocator traverse through the free lists of larger blocks until it finds a free block.
   c. The allocator then splits the free block. Whenever a free block is split into two, one block gets either used or further split, and its buddy block gets added to the corresponding free list.
   d. The 1024K block is then split into two sub-blocks, B_L and B_R, each of size 512K. The block B_L is further split and the block B_R is added to the free list of 512K blocks.
   e. The Block B_L is again split into two sub-blocks,B_{LL} and B_{LR}, each of size 256K. The block B_{LL} is further split and the block B_{LR} is added to the free list of 256K blocks. This is repeated until there is a free 128K block
   f. Note that the left block from a split is always allocated or further split and the right block is always added to the free list.
   g. In the end of step 1, we have 128K block allocated to A and also we have free blocks of sizes 128K, 256K and 512K.
3. B allocates 60K of memory.
   a. The smallest possible block that can accommodate the request is 64K.
   b. 128K block is selected and splits. The left block is assigned to B and right block -- its buddy-- is added to the free list of 64K blocks.
4. C allocates 80K of memory.
   a. The smallest possible block that can accommodate the request is 128K.
   b. There is no free block of size 128K.
   c. The 256K block is selected and splits. The left block is assigned to C and the its buddy is added to the free list of 128K blocks.
5. A ends
   a. The 128K block is freed and added to the free list. Whenever a block is freed, it can be coalesced with its buddy to form a single memory block. However, here its buddy is not free, so cannot coalesce.
6. D allocates 32K of memory.
   a. The smallest possible block that can accommodate the request is 64K(smallest possible block in the system).
   b. There is a free 64k block available and is allocated to D.
7. B ends
   a. The 64K block is freed and added to the free list. This cannot be coalesced as its buddy is occupied by D.
8. D ends
   a. The 64K block is freed. Its buddy is also free. Coalesce the blocks to form a single 128K block.
   b. The buddy of 128K block is also free, coalesce the blocks again to form a single 256K block. The buddy of 256K block is not free so further coalesce is not possible. The 256K block is added to the free list.
9. C ends
   a. The 128K block is freed. All blocks are free now. Coalesce all the buddy pairs to finally get a single free block of 1024K.

Specification

[Allocation]

void* buddy_alloc (int size);

On a memory request, the allocator returns the head of a free-list of the matching size (i.e., smallest block that satisfies the request). If the free-list of the matching block size is empty, then a larger block size will be selected. The selected (large) block is then splitted into two smaller blocks. Among the two blocks, left block will be used for allocation or be further splitted while the right block will be added to the appropriate free-list.

[Free]

void buddy_free(void *addr);

Whenever a block is freed, the allocator checks its buddy. If the buddy is free as well, then the two buddies are combined to form a bigger block. This process continues until one of the buddies is not free.

How to find a buddy and check if the buddy is free?
Suppose we have block *B1* of order *O,* we can compute the buddy using the formula below.

B2 = B1 XOR (1 << O)

We provide a convenient macro BUDDY_ADDR() for you.

Grading

10% per working test file we provide. (120% total)