Operating Systems

11. Memory Management – Part 3
Kernel Memory Allocation

Paul Krzyzanowski
Rutgers University
Spring 2015

Kernel memory

- The kernel also needs memory
  - User code calls *malloc* – kernel functions call *kmalloc*
- Lowest-level memory management: page allocator
  - Allocate and free pages (for kernel and user processes)
- But need to manage smaller chunks of memory too
  - Examples:
    semaphores, TCP connection state, network sockets, open file information, `vm_area_struct`
  - Need to minimize wasted memory
    - Page-size allocation is too wasteful
    - Reuse memory and avoid fragmentation
  - Sometimes we need physically-contiguous memory

Kernel Memory Allocation

Kernel Page Allocator

Page allocator

- With VM, processes can use non-contiguous pages
  - Memory translation makes them look contiguous
- Sometimes you need contiguous allocation
- E.g., DMA logic ignores paging
  - If we rely on DMA, we need contiguous pages

Page allocator

- Linux kernel support for contiguous buffers
- free_area: keep track of lists of free pages
  - 1st element: free single pages
  - 2nd element: free blocks of 2 contiguous pages
  - 3rd element: free blocks of 4 contiguous pages
  - …
  - 10th element: free blocks of 512 contiguous pages

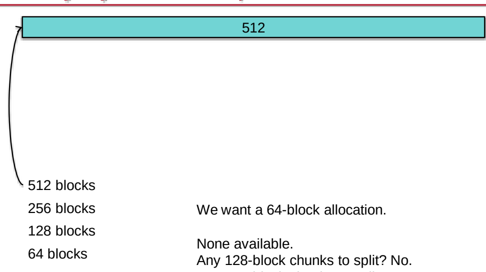
Buddy System

- Try to get the best usable allocation unit
- If not available, get the next biggest one & split
- Coalesce upon free
- Example
  - We want 8 contiguous pages
  - Do we have a block of 8? Suppose no.
  - Do we have a block of 16? Suppose no.
  - Do we have a block of 32? Suppose yes.
    - Split the 32 block into two blocks of 16. Back up.
  - Do we have a block of 16? Yes!
    - Split one of the 16 blocks into two blocks of eight. Back up.
  - Do we have a block of 8? Yes!

Buddy System: Coalescence

- When a block is freed, see if we can merge buddies
- Two blocks are buddies if:
  - They are the same size, *b*
  - They are contiguous
  - The address of the first page of the lower # block is a multiple of *2b × page_size*
- If two blocks are buddies, they are merged
- Repeat the process.

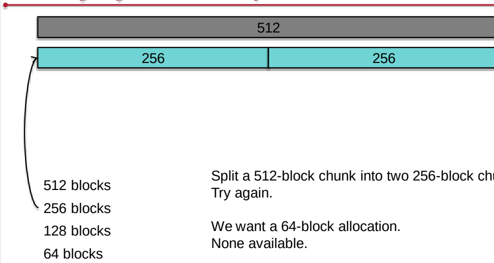
Buddy System Example



We want a 64-block allocation.

None available.
Any 128-block chunks to split? No.
Any 256-block chunks to split? No.
Any 512-block chunks to split? Yes.

Buddy System Example

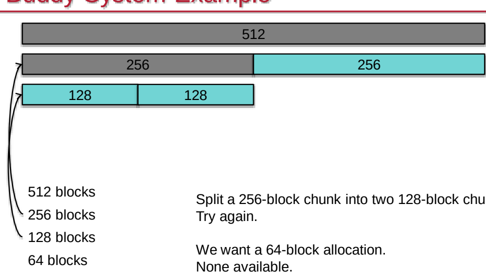


Split a 512-block chunk into two 256-block chunks.
Try again.

We want a 64-block allocation.
None available.

Any 128-block chunks to split? No.
Any 256-block chunks to split? Yes.

Buddy System Example

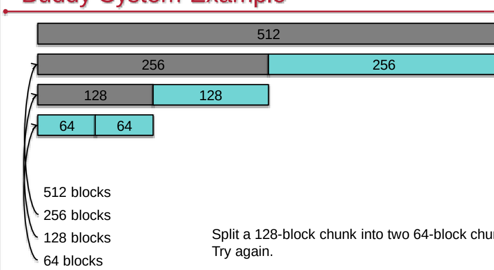


Split a 256-block chunk into two 128-block chunks.
Try again.

We want a 64-block allocation.
None available.

Any 128-block chunks to split? Yes.

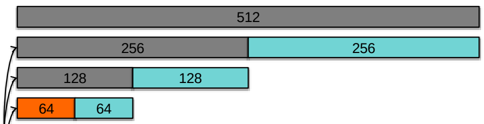
Buddy System Example



Split a 128-block chunk into two 64-block chunks.
Try again.

We want a 64-block allocation.
Got it!

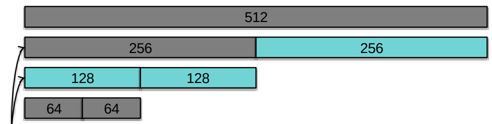
Buddy System Example



512 blocks
256 blocks
128 blocks
64 blocks

Requester gets the 64-block chunk.
Later, it is no longer needed and is returned.

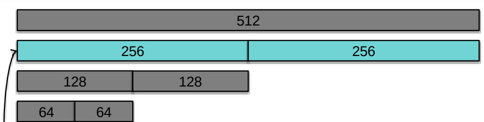
Buddy System Example



512 blocks
256 blocks
128 blocks
64 blocks

Combine (coalesce) the two 64-block chunks into a128-block chunk

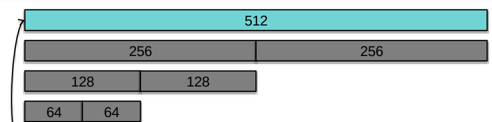
Buddy System Example



512 blocks
256 blocks
128 blocks
64 blocks

Combine the two 128-block chunks into a 256-block chunk

Buddy System Example



512 blocks
256 blocks
128 blocks
64 blocks

Combine the two 256-block chunks into a 512-block chunk

Linux Memory Management

Buddy System: page allocation

- Used for *pages* not *objects*
- Manages lists of physically contiguous memory pages
  - Buddy system used within each zone
  - E.g., allocate 8 contiguous pages of DMA-capable DRAM
- Maps regions of memory to MMU page tables
  - Regions are multiples of 2^x pages
- Lists of
  - 1 page blocks
  - 2 page blocks
  - 4 page blocks
  - 2^n page blocks (n defined by MAX_ORDER constant)

Zone Allocator: memory specification

- Ranges of pages may have different properties
  - E.g., some architectures allow peripherals to perform DMA only for addresses < 16 MB.
- All memory is divided into zones
  - DMA: memory accessible for DMA
  - NORMAL
  - HIGHMEM: for system use (file system buffers, user space, etc.)
- Allocation is handled per zone. An allocation request specifies zones in most- to least-preferred order

This is not a memory allocator but a way of qualifying specific page needs

Slab Allocator: object allocation

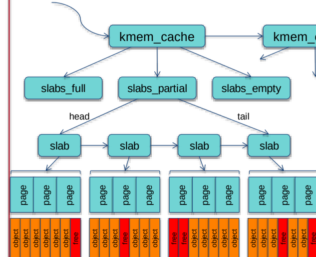
- Kernels often allocate specific objects, not arbitrary sizes
- Initializing an object sometimes takes more time than allocating it
  - If possible, keep object initialized (e.g., call *mutex_init* just once)
  - Bring object back to its initial state at deallocation
- Key concept
  Pre-allocate caches of contiguous memory to make it efficient to allocate allocation requests for objects of a specific size.

Slab Allocator: components

- Terms
  - Object: requested unit of allocation
  - Slab: block of contiguous memory (often several pages)
    - Each slab caches similarly-sized objects
    - Avoids fragmentation problems
  - Cache: storage for a group of slabs for a specific object
    Each unique object type gets a separate cache
- Slab states
  - Empty all objects in the slab are marked as free
    - The slab can be reclaimed by the OS for other purposes
  - Full: all objects in the slab are marked as in-use
  - Partial: the slab contains free and in-use objects

Slab allocator structure

cache_chain: linked list of slab caches



- Try to find a free location for the requested object on a partial slab in a cache for that type of object.
- If not found, then allocate a new slab and assign it to a cache

Slab allocator: operations

- kmem_cache_create: create a new cache
  - Typically used when the kernel initializes or a kernel module is loaded
  - Identifies the name of the cache and size of its objects
  - Separate caches for inodes, directory entries, TCP sockets, etc.
- kmem_cache_destroy: destroy a cache
  - Typically called by a module when it is unloaded
- kmem_cache_alloc: allocate an object from a named cache
  - cache_alloc_refill may be called to add memory to the cache
- kmalloc / kfree: no object (cache) specified
  - Iterate through available caches and find one that can satisfy the size request

Slab allocator: advantages

- Memory always gets allocated in the size requested
- No internal fragmentation
- Quick allocation

SLOB: Simple List Of Blocks

- Alternative memory allocator to Slab
- Designed for small and embedded memory-constrained systems
- Heap allocator
  - SLOB heap = three singly linked list of pages
    1. small objects (< 256 bytes)
    2. medium (< 1024 bytes)
    3. large (< PAGE_SIZE)
    - Lists are grown on demand with calls to `__get_free_page`
  - Blocks < page size returned from the heap
    - Return 8-byte aligned block
  - All blocks, allocated & free contain a header (metadata)
    - Size of this block and offset of next free/allocated block
  - Bytes ≥ PAGE_SIZE
    - kmalloc calls `__get_free_pages` directly and keeps a linked list of allocated pages

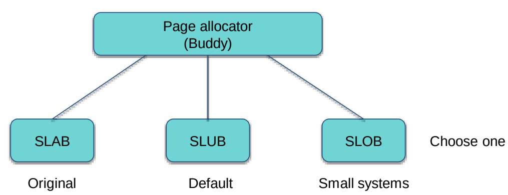
SLOB: Simple List Of Blocks

- Uses a first-fit allocation algorithm
- Suffers from fragmentation

SLUB allocator

- Current default kernel memory allocator on Linux
- Similar to SLAB: same slab structures
  - Reduced performance overhead
  - Support arbitrary number of CPUs and arbitrary # of slabs
  - Metadata that was stored per slab moved to the page structure (info that the kernel uses to keep track of each page)
  - Per-CPU queues removed to improve performance with multiprocessor systems

Linux kernel memory allocation



The End