Memory Management III
Memory Allocation

COMS W4118

Prof. Kaustubh R. Joshi

krj@cs.columbia.edu

http://www.cs.columbia.edu/~krj/os

References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s

Outline

- Dynamic memory allocation overview
- Heap allocation strategies

Dynamic memory allocation

- Paging solves contiguous memory problem
  - Virtual memory is contiguous
  - Pages can be discontiguous
- But, paging doesn't always work for kernel memory
  - Requests smaller than a page (e.g., kmalloc)
  - DMA hardware doesn't understand paging
    - unless IOMMU support is available
- Two ways of dynamic allocation
  - Stack allocation
    - Restricted, but simple and efficient
  - Heap allocation
    - More general, but less efficient
    - More difficult to implement

Dynamic allocation issue: fragmentation

- Fragment: small chunk of free memory, too small for future allocation requests ("holes")
  - External fragment: visible to allocation system
  - Internal fragment: visible to process (e.g. if allocate at some granularity)

- Goal
  - Reduce number of holes
  - Keep holes large

- Stack fragmentation v.s. heap fragmentation
  - Stack: all free space is one big hole – no fragmentation
  - Can only deallocate when everything above you is gone
  - Heap: fragmentation possible

Typical heap implementation

- Data structure: free list
  - Chains free blocks together
- Allocation
  - Choose block large enough for request
  - Update free list
- Free
  - Add block back to list
  - Merge adjacent free blocks (reduce fragmentation)

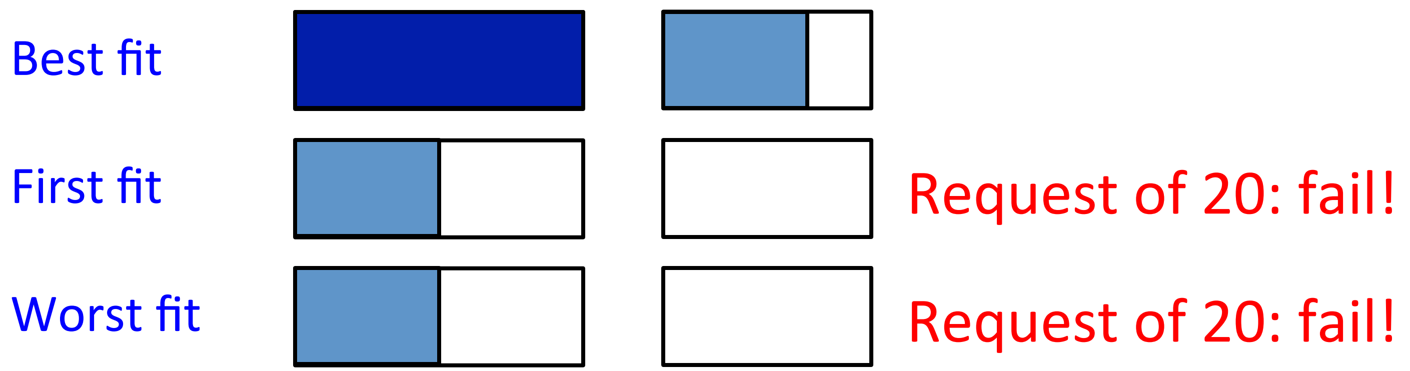
Heap allocation strategies

- Best fit
  - Search the whole list on each allocation
  - Choose the smallest block that can satisfy request
  - Can stop search if exact match found
- First fit
  - Choose first block that can satisfy request
- Worst fit
  - Choose largest block (most leftover space)

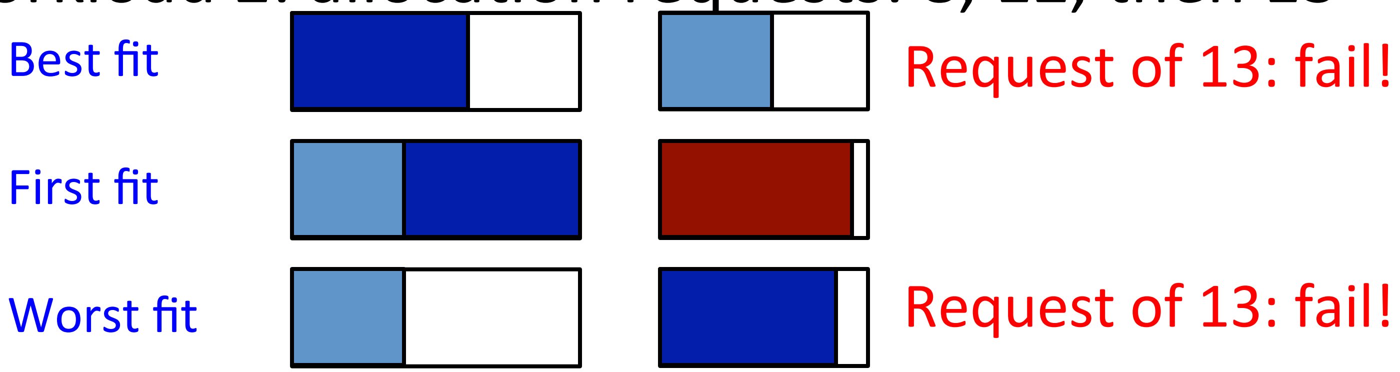
Which is better?

Example

- Free space: 2 blocks, size 20 and 15
- Workload 1: allocation requests: 10 then 20



- Workload 2: allocation requests: 8, 12, then 13



Comparison of allocation strategies

- Best fit
  - Tends to leave very large holes and very small holes
  - Disadvantage: very small holes may be useless
- First fit:
  - Tends to leave "average" size holes
  - Advantage: faster than best fit
- Worst fit:
  - Simulation shows that worst fit is worst in terms of storage utilization

Buddy allocator motivation

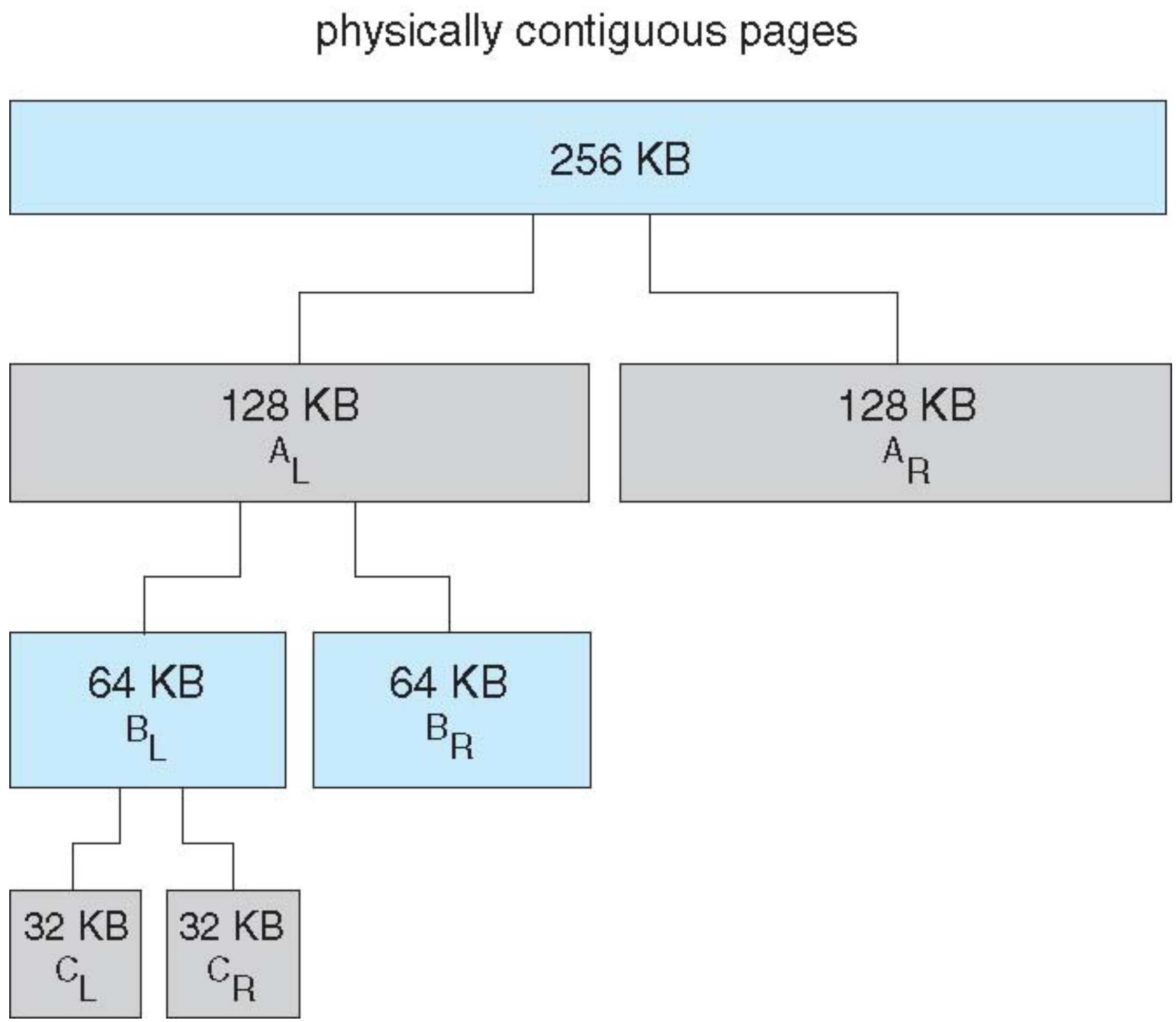
- Allocation requests: frequently 2^n
  - E.g., allocation physical pages in FreeBSD and Linux
  - Generic allocation strategies: overly generic
- Fast search (allocate) and merge (free)
  - Avoid iterating through entire free list
- Avoid external fragmentation for req of 2^n; keep free pages contiguous

Real: used in FreeBSD and Linux

Buddy allocator implementation

- Allocation restrictions: 2^k, $0 <= k <= N$
- Data structure
  - N free lists of blocks of size $2^0, 2^1, \ldots, 2^N$
- Allocation of 2^k:
  - Search free lists (k, k+1, k+2, …) for appropriate size
    - Recursively divide larger blocks until reach block of correct size
    - Insert “buddy” blocks into free lists
- Free
  - Recursively coalesce block with buddy if buddy free

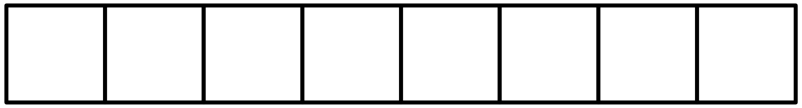
Buddy System Allocator



Buddy allocation example

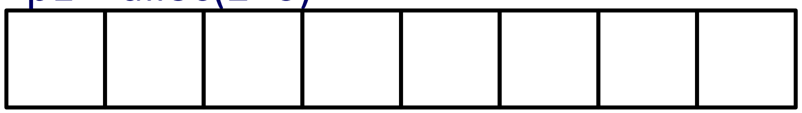
Color Legend:
Black: allocated.
Other: on freelist of that color.

freelist[3] = free list for blocks of 2^3 pages.



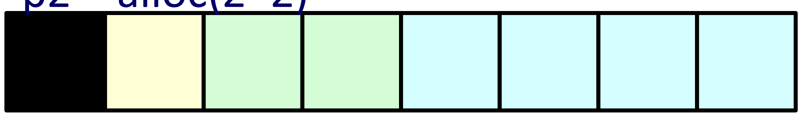
freelist[3] = {0}

p1 = alloc(2^0)



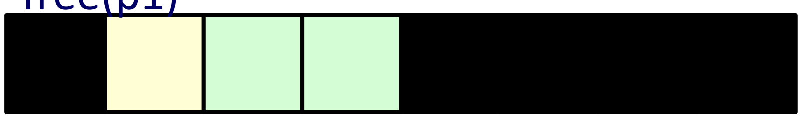
freelist[0] = {1}, freelist[1] = {2}
freelist[2] = {4}

p2 = alloc(2^2)



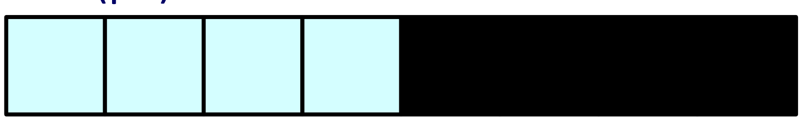
freelist[0] = {1}, freelist[1] = {2}

free(p1)



freelist[2] = {0}

free(p2)



freelist[3] = {0}

Pros and cons of buddy allocator

- Advantages
  - Fast and simple compared to general dynamic memory allocation
  - Avoid external fragmentation by keeping free physical pages contiguous
- Disadvantages
  - Internal fragmentation
    - Allocation of block of k pages when k != 2^n

Slab allocator

- Motivation:
  - Frequent (de)allocation of some kernel objects, E.g., task_struct, inode
  - Other allocators: overly general; assume variable size
- Cache: slab of "slots"
  - Each cache holds only single object type (task_struct, inode, dentry, vma)
  - Each cache has one (or more) slabs, each 1 page long
  - Each slab is split into slots
  - Slot size = object size
- Slab operations
  - Free memory management = bitmap
  - Allocate: set bit and return slot
  - Free: clear bit
- Used in FreeBSD and Linux on top of buddy page allocator
  - For objects smaller than a page
  - kmem_cache_create: create a new cache for your own object type
  - kmem_cache_alloc: allocate new object from cache

Slab Allocation