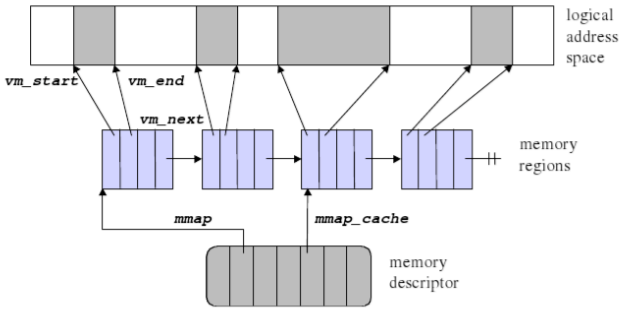
Dynamic Memory Management

- The Linux Perspective
- Allocating memory: The Interface
- Buddy System
- Slab Allocation

- *Reading: Silberschatz (8th ed.), Chapters 9.8 and 21.6*

Memory Areas

Memory areas (regions) are intervals of legal addresses.



The Process Address Space

- When does new memory get allocated?
  - Process stack grows
  - Process "creates" (attaches) to shared memory segment (`shmat()`)
  - Process expands heap (`malloc()`)
  - New process gets created (`fork()`)
  - New program gets loaded into memory (`execve()`)
  - Map a file to memory (`mmap()`)
- Create new address interval:
  - Kernel uses `do_mmap()` call.
  - Available through system call `mmap()` in user space.

Allocating Pages

- Requesting frames:

```
struct page * alloc_pages(uint gfp_mask, uint order)
```

- Requesting pages (logical addresses):

```
ulong __get_free_pages(uint gfp_mask, uint order)
```

- In both cases:
  - request allocates 2^{order} pages/frames
  - `gfp_mask` specifies details about request:
    - memory zone
    - behavior of allocator (blocking/unblocking request, etc.)
    - e.g. `GFP_KERNEL`, `GFP_ATOMIC`, `GFP_DMA`, etc.

Allocation at Different Levels

- `alloc_pages()` and `__get_free_pages()`
  - allocate pages, at low level
  - useful to allocate contiguous pages/frames.
- byte-sized allocations:
  - `kmalloc(size, gfp_mask)`
    - allocate physically contiguous sequence of bytes
  - `vmalloc(size, gfp_mask)`
    - allocate virtually contiguous sequence of bytes
- explicit user-level allocation:
  - `malloc(size)`
    - allocate virtually contiguous sequence of bytes at user level

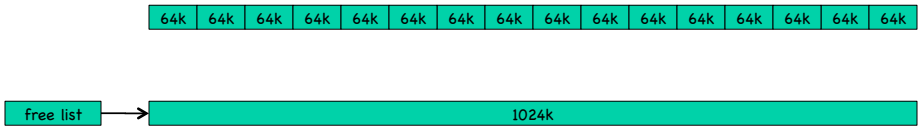
How does this all work?

- `alloc_pages()` and `__get_free_pages()`
  - allocate pages, at low level
  - useful to allocate contiguous pages/frames.

  } Buddy System!
- byte-sized allocations:
  - `kmalloc(size, gfp_mask)`
    - allocate physically contiguous sequence of bytes
  - `vmalloc(size, gfp_mask)`
    - allocate virtually contiguous sequence of bytes

  } Slab Allocator (+ caching)
- explicit user-level allocation:
  - `malloc(size)`
    - allocate virtually contiguous sequence of bytes at user level

Naïve Allocation in Action



Buddy System Allocation

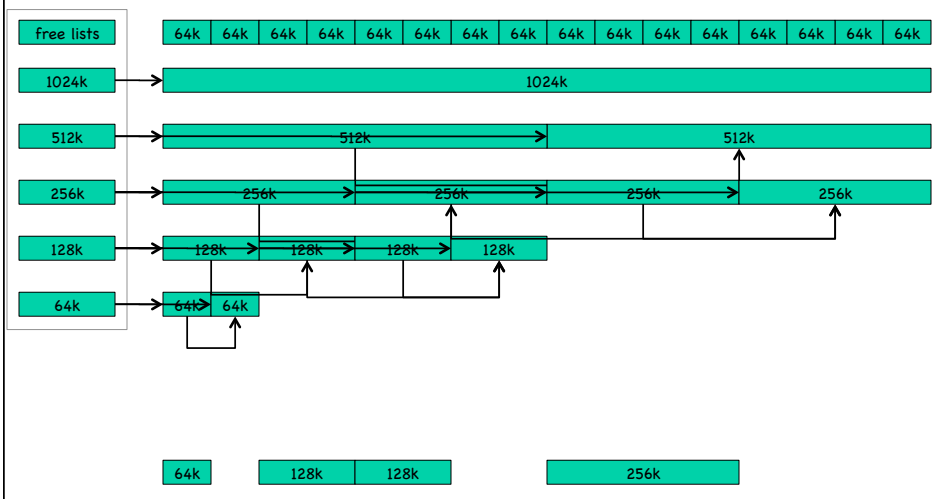
Harry Markowitz
1927-
1990 Nobel Memorial Prize in Economics

- Allocation:
  - Increase size of request to next power of 2*.
  - Look up block in free lists.
  - If exists, allocate.
  - If none exists, split next larger block in half, put first half (the "buddy") on free list, and return second half.
- De-Allocation:
  - Return segment to free list.
  - Check if buddy is free. If so, coalesce.
- For details, see lecture.

(*) For case of binary buddy system.

References: Donald Knuth: The Art of Computer Programming Volume 1: Fundamental Algorithms. Second Edition (Reading, Massachusetts: Addison-Wesley, 1997), pp. 435-455. ISBN 0-201-89683-4

Buddy System in Action



Slab Allocation

- First described by Jeff Bonwick for the SunOS kernel.
- Currently used in Linux and other kernels.
- Key observations:
  - Kernel memory often used for allocated for a finite set of objects, such as file descriptors and other common structures.
  - Amount of time required to initialize a regular object in the kernel exceedes the amount of time required to allocate and de-allocate it.
- Conclusion:
  - Instead of freeing the memory back to a global pool, have the memory remain initialized for its intended purpose.
- References: "The Slab Allocator: An Object-Caching Kernel Memory Allocator (1994)"

Slab Allocation (II)

- Set of objects pre-allocated
- Marked as free
- When needed, assign a free one and mark as used
- No free ones available?
  - allocate a new slab
  - slab states (full, empty, partial)
  - fill partial slab first
- Advantages:
  - no fragmentation
  - memory requests satisfied quickly