IE 411: Operating Systems

Free space management

Announcements

- Project 1 is out and due on March 14
- Midterm 2 in two weeks

Last lecture

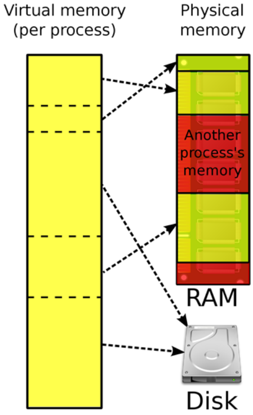
- Before writing a flash page (small chunk of data):
    - First must erase the flash block (large chunk of data) where the page lives
    - This takes a relatively long time

Last lecture

- Before writing a flash page (small chunk of data):
  - First must erase the flash block (large chunk of data) where the page lives
  - This takes a relatively long time
- Writing a page too often will cause it to wear out
  - 10,000 to 100,000 writes to a page
  - Page is no longer usable

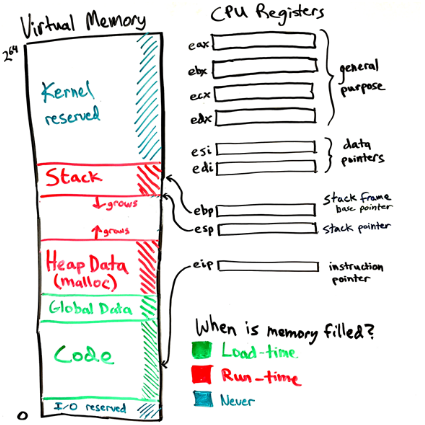
Virtual memory

- Splits memory between various processes
- But gives each process the illusion that it has the full address space
- Code uses virtual memory addresses (called "logical address")
- CPU somehow translates to physical addresses assigned to that process



The process' view of Memory

- Code and global data are filled by exec syscall to load a program
- A new frame is pushed on the stack whenever a function is called. (And popped on return.)
- Heap data is managed by malloc
- How does malloc work?

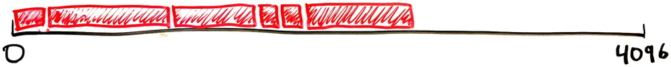


Free-space management

- Given: A single block of contiguous memory
- Goals: Handle malloc and free requests
  - void* malloc(size_t n): find a unused block of size n
  - void free(void* ptr): reclaim a block that was previously malloc-ed
- Minimize the total extent of memory required (be compact)
- Minimize the time required to malloc and free

Heap dynamics

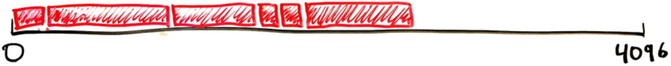
- Ideally, malloc would waste no space:



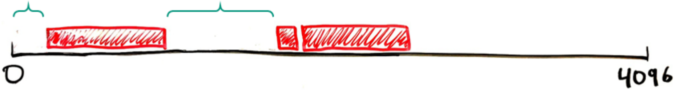
- But malloc is used for dynamic memory
    - it will be freed later, at some unknown time

Heap dynamics

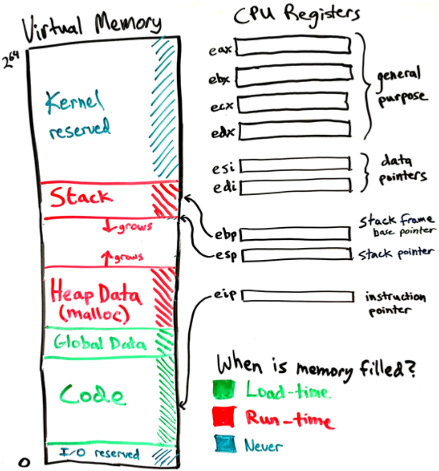
- Ideally, malloc would waste no space:



- But malloc is used for dynamic memory
  - it will be freed later, at some unknown time
- Frees create vacancies in the Heap that waste space, but can be re-allocated



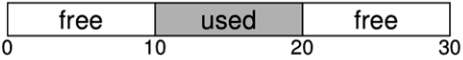
Grow the Heap only as a last resort



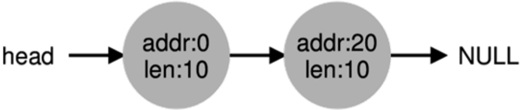
- Goal is to make maximum use of the Heap range already in use
- Look for a free block $\geq$ the current malloc request
- If none is found, then we have to expand the heap
- Memory leaks cause heap to grow indefinitely (if you forget to free)

Free list is a linked list tracking free blocks

- 30-byte heap



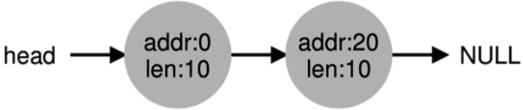
- Free list



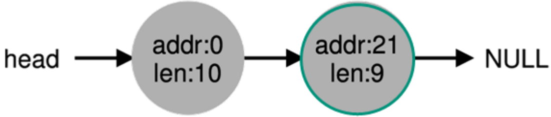
- Linked lists are just one way to track free space

Allocate memory by splitting free blocks

- Before



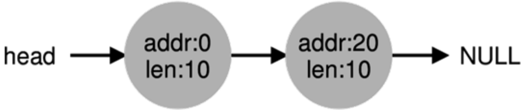
- After allocating one byte from the second free block



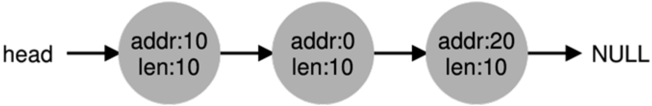
- The policy decides which block to choose

Coalescing eliminates artificial fragmentation

- Before



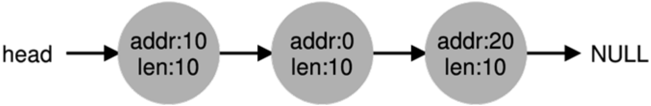
- After freeing 10 bytes of data at address 10
  - We usually add new nodes to a linked list's head



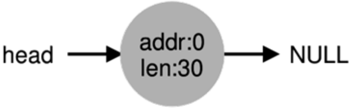
  - This is a large contiguous free block

Coalescing eliminates artificial fragmentation

- Before

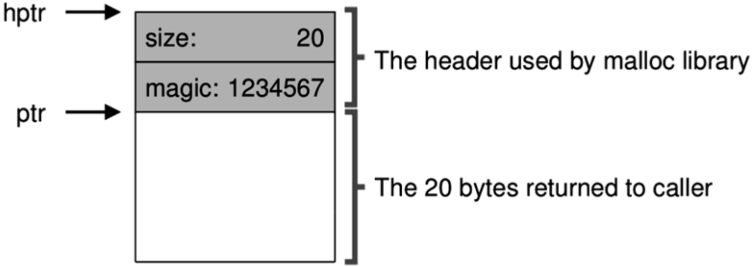


- After coalescing (joining of adjacent free blocks)

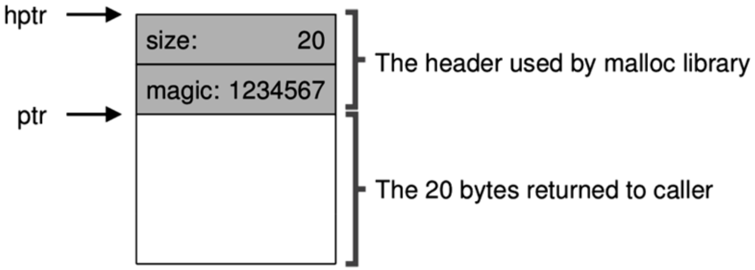


A trick to help with frees

- free(ptr) doesn't tell us how long the block is, just where it starts
  - But we need that information to free the block
- Solution: cleverly prefix the block with a header
  - e.g., malloc(20)


hptr
ptr
size: 20
magic: 1234567
The header used by malloc library
The 20 bytes returned to caller

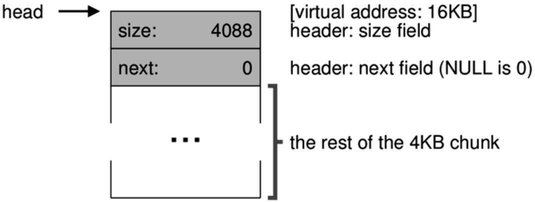
A trick to help with frees


hptr
ptr
size: 20
magic: 1234567
The header used by malloc library
The 20 bytes returned to caller

- To handle free(ptr) just look at (ptr – sizeof(header_t)) to find the block size
- What's up with the magic number? (more on this later)

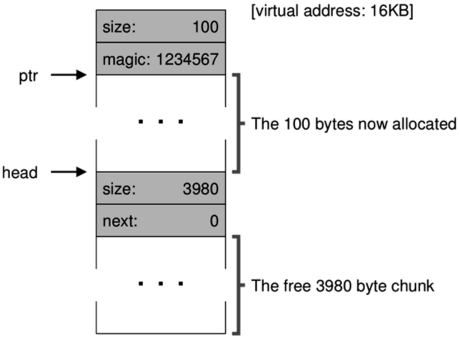
Example

- We setup a free list in the Heap
- Size is 4K (little less due to the header)
- head is the pointer for the free list



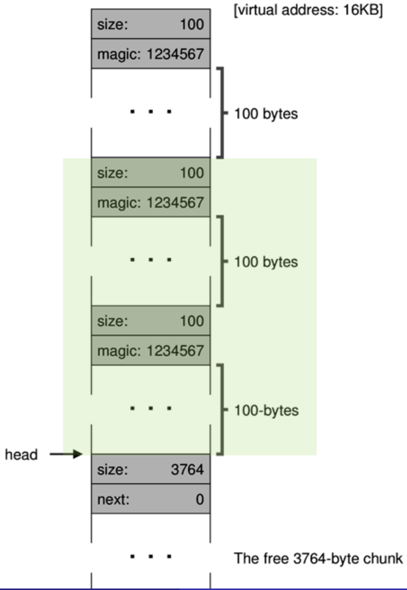
Example

- A program mallocs 100 bytes worth of data
- 8 extra bytes for the header
- We split the free space and update the size of the free list header

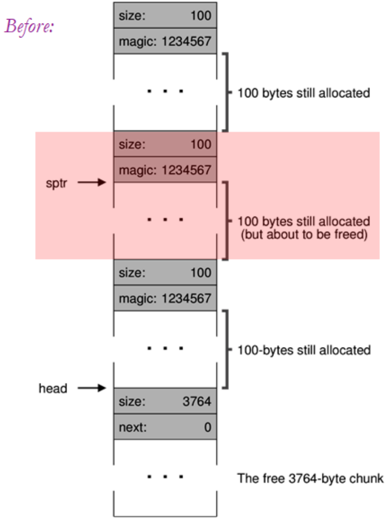


Example

- Two more requests for 100 bytes are made
- Head pointer is updated

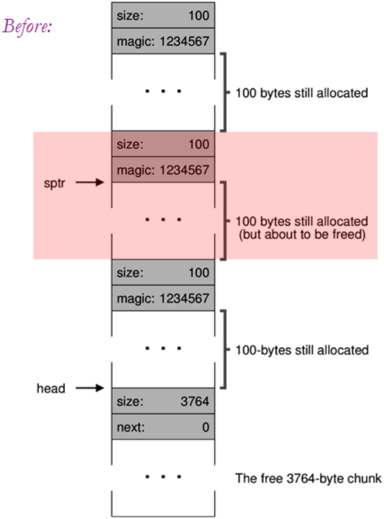


Example



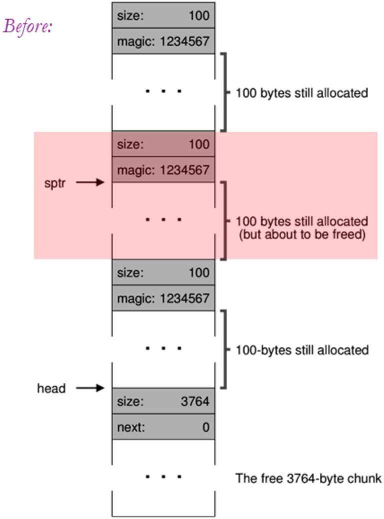
- Look back from sptr to find that size=100
- To free the block, just two small changes are needed

Example



- Look back from sptr to find that size=100
- To free the block, just two small changes are needed
  1. sptr - 4 = head

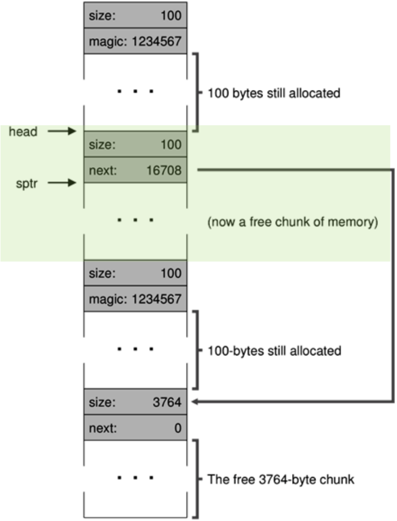
Example



- Look back from sptr to find that size=100
- To free the block, just two small changes are needed
  1. sptr - 4 = head
  2. head = sptr - 8

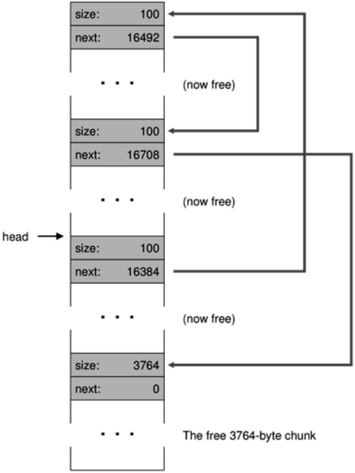
Example

- Net change
  - Head is moved to reference that newly free 100 bytes
  - The next free space pointer is updated
- Middle chunk is freed



Example

- As the remaining memory is freed, the list pointers are updated
- Coalescing the free spaces is necessary in the free list to restore the heap's actual capacity



Magic number

- It's just an unusual, large numeric constant (always the same number)
- It allows free to detect whether the pointer it received is valid
- If there is no magic number behind the pointer, then free should abort and warn the user

Magic number

- It's just an unusual, large numeric constant (always the same number)
- It allows free to detect whether the pointer it received is valid
- If there is no magic number behind the pointer, then free should abort and warn the user
  - Maybe the code already called free? a "double free" error

Magic number

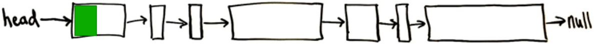
- It's just an unusual, large numeric constant (always the same number)
- It allows free to detect whether the pointer it received is valid
- If there is no magic number behind the pointer, then free should abort and warn the user
  - Maybe the code already called free? a "double free" error
  - What could happen if we allowed a double free to proceed?

Choosing a free block

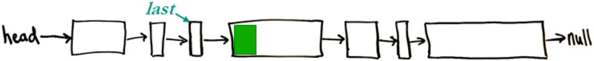
- We have seen a basic malloc/free mechanism
- There are still some policy decisions to make:
  - Which of the free blocks do we choose for a given allocation?
  - When do we coalesce?

Choosing a free block

- First fit: simple and fast

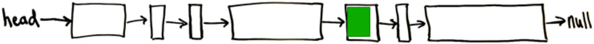


- Next fit: start looking where you left off

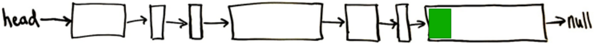


Choosing a free block

- Best fit: try to leave the smallest remainder
  - but have to search the whole list and leaves small holes (hard to reuse)

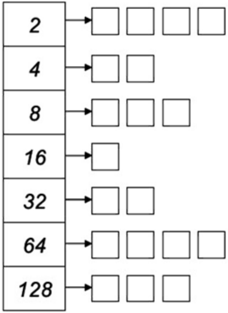


- Worst fit: try to leave large remainders that are easy to use



Segregated lists (Slab memory allocator)

- Instead of keeping one list of free blocks, we can keep different lists for small, medium, and large blocks
- This will allow us to find the right sized block more efficiently
- Just keep an array of free lists (many heads)
- On free (or if splitting), add free block to the appropriate list
- Many variations are possible, but this design often uses fixed-length chunks (powers of 2)


2
4
8
16
32
64
128