K-OS Memory Management

Kai Chen

CS134c Spring 2003

Overview

- A data structure to keep track of available memory. (in pages)
  - Bitmap : one bit for each page
  - Stacks : pushes on free pages
- Virtual Memory with Paging
  - Page directory, page tables, page frames ...

Paging Basics

- **Page Boundary**
  - Block of memory 4Kb aligned. (starting at address where the lower 12 bits are 0)
- **Page Directory**
  - An array of 4-byte page table specifiers. (up to 4Kb)
- **Page Table**
  - An array of 4-byte page specifiers. One page table maps 4Mb memory and takes up to 4Kb to store.
- **Page Frame**
  - 4Kb of contiguous memory. Starting at page boundary.
- **Page directory and page table entries**
  - Where a page points to. And page attributes.

Page Directory/Table Entry

31 ... 12	11...9	8 ... 7	6	5	4 ...3	2	1	0
address	Avail	Reserved	D	A	Reserved	U/S	R/W	P

- Address: (physical) 20 bits is enough. Last 12 bits always 0
- Avail: Can be used however we want
- D: dirty bit
- A: accessed bit
- U/S: user/superuser
- R/W: read/read and write
- P: present

Setting up Paging (example)

```
Unsigned long * page_dir = (unsigned long *) 0x9C000;
unsigned long * page_table = (unsigned long *) 0x9D000;
unsigned long addr = 0;

for(i=0; i<1024; i++){
      page_table[i] = addr |  3;
      addr += 4096;
}
page_dir[0] = page_table | 3;

for(i=0; i<1024; i++){
      page_dir[i] = 0 | 2;
}
```

Enabling Paging

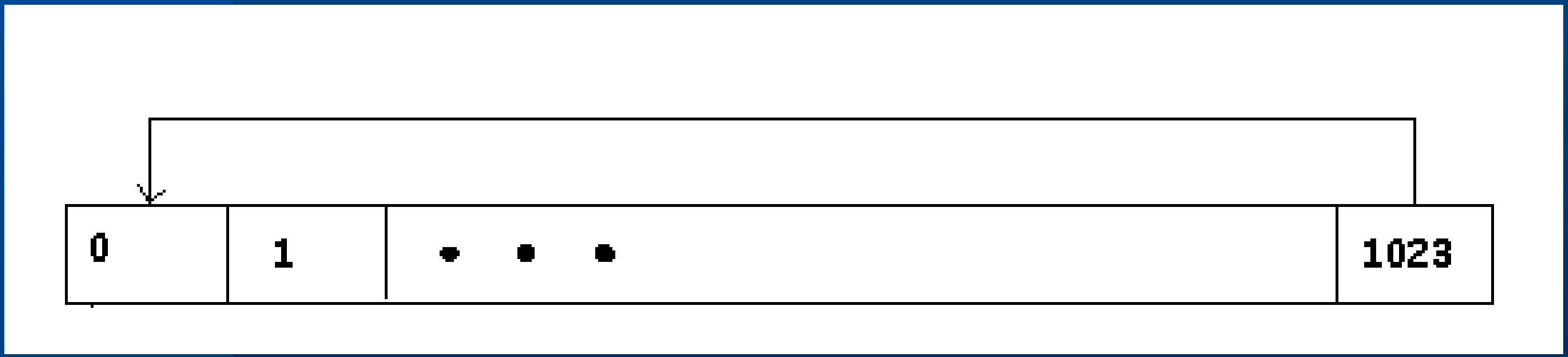
- Put the address of the page directory in CR3.
- Set bit 31 of CR0 to enable paging.

Virtual Memory

- **Physical Memory Organization**
  - **Physical pages**
  - **Page usage count**
- **Virtual Memory Organization**
  - **Kernel Page Directory: 0x70000**
  - **Kernel Page Tables: 0x71000**

Self Reference

- The last entry of the page directory points to itself.
- We can refer to page directory and page tables via virtual memory:
  - PAGE_SELF = 1023
  - PAGE_DIR_VADDR = (PAGE_SELF<<22) | (PAGE_SELF<<12)
  - PAGE_TABLE_VADDR = (PAGE_SELF << 22)



VM Components

- Page Allocator:
  - Find a free page in page_use_count vector
  - Return the physical address
  - Deallocation is the natural opposite
- Memory Map (mmap(vaddr, paddr, attr)):
  - Compute PDE and PTE for vaddr:
    - PDE(vaddr) = vaddr >> 22
    - PTE(vaddr) = vaddr >> 12
  - Page align paddr
  - Insert paddr into the page table

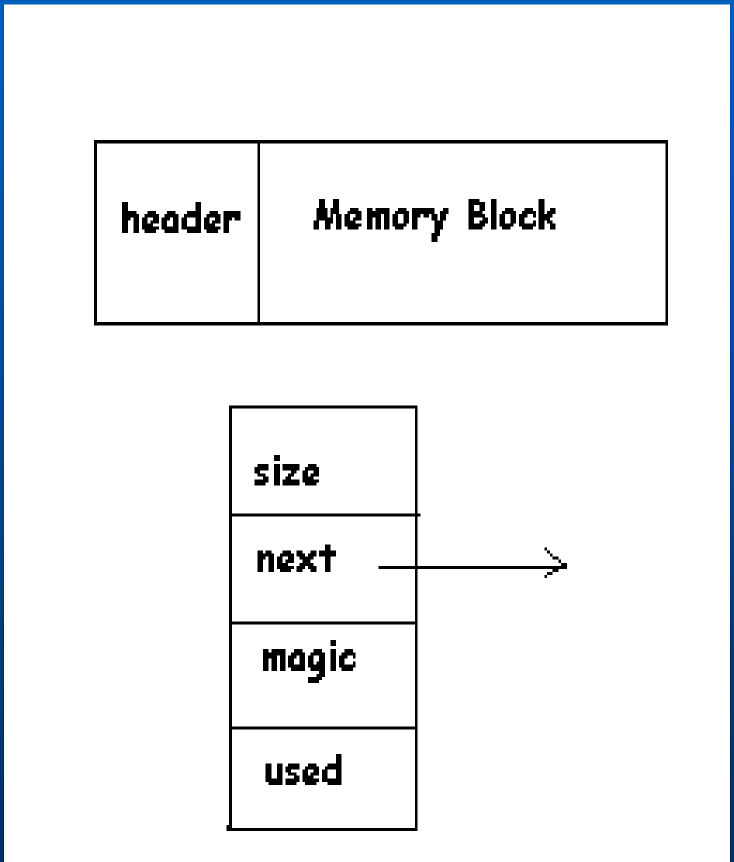
VM Components (Continued)

- **Low Level Allocation (void *morecore):**
  - **Allocation done in pages (4Kb aligned)**
  - **Use alloc_page() to allocate physical pages**
  - **Use mmap() to map pages to virtual address**
  - **Return the start of the virtual address**
  - **Kernel Heap starts at 0x100000**
  - **We do not allow down sizing**

VM Components (Continued)

- High Level Allocation (kmalloc(size_t size))
  - Organize memory in blocks.
  - Each block has a header:

```
struct header {
        size_t size;
        header *next;
        unsigned magic : 31
        unsigned used : 1
}
```



VM Components (Continued)

- High Level Allocation: kmalloc (size_t size)
  - Go through the kernel heap to find a large enough free block.
  - If found:
    - split it into 2 and save the extra
    - return the block with right size
  - If not found:
    - use morecore() to allocate a new block
    - add to kernel heap
    - split to save the extra
    - return the block with right size

VM Components (Continued)

- **High Level Deallocation: kfree()**
  - **sanity check: consistent magic number**
  - **find the block in the heap**
    - **return error if not found**
  - **mark the block as free**
  - **combine the adjacent free blocks to reduce fragmentation**

Summary

- **Allocation:**
  - allocate physical page
  - map to virtual memory
  - allocate virtual memory in 4Kb blocks
  - reorganize into linked list of finer blocks
- **Deallocation:**
  - mark block as free
  - combine adjacent free blocks