

Pintos Virtual Memory Management Project

(CS3204 Spring 2006 VT)

Yi Ma

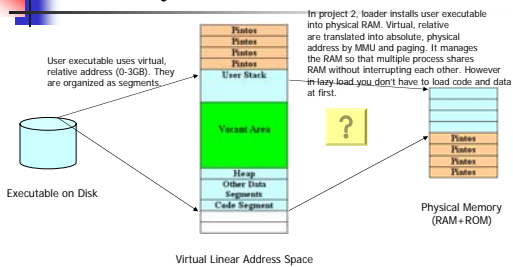
1

Outline

- Memory and Access
- Current Status in Pintos
- Requirements
- Suggestion

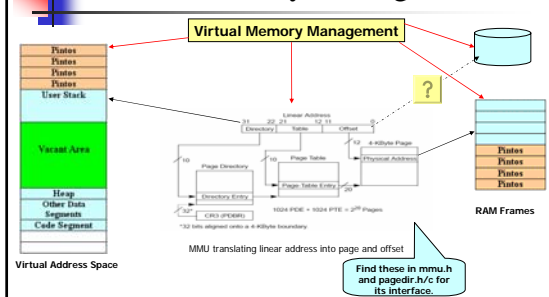
2

Memory



3

Virtual Memory Management



4

Current Status (after a successful project2)

- Kernel's memory has been well handled.
- MMU—PD, PT, pagerdir—address maps.
- Per-process page table (process.c).
- Preload data and code segment and stack (load, load_segment, and setup_stack).
- Fixed stack and limited size and number of programs.

5

Requirement Overview

- Page Table Management
 - Address mapping and page fault
- Swapping
 - Choose a replacement alg., like a 1-bit clock alg. (NRU)
- Lazy loading
 - Load on demand
- Memory mapped files

6

Page Table Management

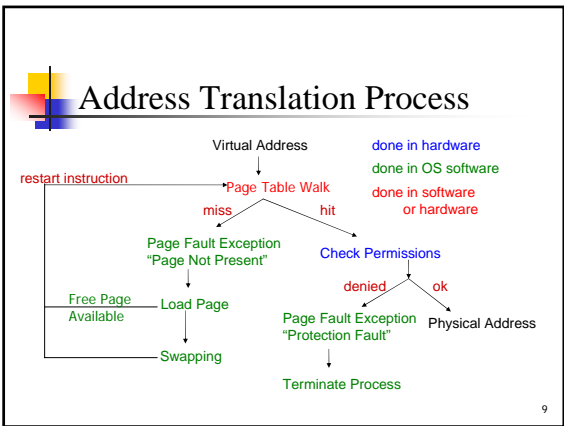
- Functionalities
 - Virtual mapped to physical
 - Locate page (RAM or SWAP)
 - Physical back to virtual (evicting)
 - This vs. PD & PT?
- Virtual vs. Physical Page
 - Install virtual page into a physical frame
 - Mapping through MMU's PD and PT
 - Keep track

7

Page Fault

- Pre-project 3
 - Install new PD when switching
 - Setup all mapping
 - Fault on unmapped address and kill
- Post-project 3
 - Page_fault is no longer always an error
 - May have to swap in or load
- How
 - Get the faulting address
 - Find the page (file or swap)
 - Let it in (may require eviction of other page currently located in the frame.)
 - Update PTE

8



Confusing?

- Too many pages and tables?
- MMU: pde, pte
- Frame page
- Virtual page
- Swapped page
- Their cross links

10

Frames

- Only user pool
- ram_pages in load.s and init.c?
- It's relation with current palloc.
- Two ways for your frame allocator.
- Where you initialize.

11

Virtual Pages

- It is the core of the virtual memory.
- Relation with user address space. (continuous or not)
- Relation with frames
- Relation with swap disk
- Relation with executable file
- Relation with MMU's PD and PT

12

Pages on disk

- Memory in 4KB pages
- Disk in 512B sectors
- How to organize the swap disk so that it could best conform page system
- Design data structure to implement the above design

13

What is the problem

- Starting/end address of a page
- Permission of the page
- Location of the “virtual” page: RAM, SWAP, or File
- Keep track: when and what to update

14

Stack Growth

- First page allocated in setup_stack
- Look at memory layout: there is a vacant between heap and stack
- i386 “push” at most 32B to the stack at once
- Stack is growing downwards
- Catch the stack pointer—esp
- When: page_fault
- Limit the size of stack reasonably

15

Swapping

- Loaded pages got evicted—swap out
- Evicted pages got reloaded—swap in
- Swap out on page fault and no free frame
- Swap in on page fault
- Manage Disk (track occupied or free sectors)
- No order
- Parallel

16

Evict a page

- Keep track where is the free frames
- Keep track who are evict-able frames
- Pick a replacement algorithm like 1-bit alg.
- Accessed/dirty bit in PTE
- Send it to swap
- Update PD & PT

17

Lazy Loading

- Now: load all loadable pages into RAM
- Lazy: load these only when needed
- Not swapping, use the executable file
- Keep track the location

18

Memory mapped files

- Keep a copy of an open file in memory
- Keep it in continuous virtual pages
- File size is not multiple of PGSIZE—stick-out, cause partial page
- Mapped IDs
- Don't map when: zero address or length, overlap, or console
- Address or pages are "virtual"
- System calls: mmap(fd, addr), munmap(ID)

19

On process termination

- Clean you page table
- Free your frames
- Free your SWAP
- Close all files

20

Other issues

- Access user data
 - In project 2, need only verify user address
 - In project 3, need handle actual access to the content of user memory
 - Need protect: check address, lock frame, read/write, and unlock.
- Synchronization
 - Allow parallelism of multiple processes
 - Page fault form multiple processes
 - E.g., A's page fault need I/O (swap, lazy load); B's page fault need not, then B should go ahead;
- Containers
 - Proper container will affect your design significantly
 - Bit map, hash, list, and array
 - How many copies?
 - Make it simple

21

Suggested Order

- Frame table management: your own "palloc"—modify or design new
- Switch to your allocator—modify process.c
- Virtual page table management
- Page Fault
- Stack growth, mmap, and reclamation
- Swap: accessed/dirty bits, alias, parallelism, and eviction algorithm
- <http://courses.cs.vt.edu/~cs3204/spring2006/wmcquain/index.html>

22

Other suggestions

- Go over your lecture notes
- Design before start
- Keep an eye on Dr. Back's project pages

23