Virtual Memory

DON’T PANIC!

Aaron Blankstein
Scheduling Administrivia

• Design Reviews on Monday!
  – Signup is live?
  – Show up prepared

• Project due the following Sunday.
What’s going to happen in this Project

• Different memory layouts for different tasks
• Restriction of user processes to the user mode
• You will use usb for swap storage
  – Process uses whatever location it was originally loaded from (swap_loc)
  – Multiple process instances will be broken this means
What You’ll Implement

• Initializing Memory (kernel page stuff)
• Setting up each process’ memory
• Handling Page Faults
• Swap in and Swap out
• Extra Credit: better eviction policy
2-Level Page Table (i386)

- Link on the Project Description to Intel Manual

*32 bits aligned onto a 4-KByte boundary.
Directory Entries

![Diagram of a directory entry]

- Page-Table Base Address
- Available
- Global page (Ignored)
- Page size (0 indicates 4 KBytes)
- Reserved (set to 0)
- Accessed
- Cache disabled
- Write-through
- User/Supervisor
- Read/Write
- Present
| Available for system programmer’s use | Global Page | Page Table Attribute Index | Dirty | Accessed | Cache Disabled | Write-Through | User/Supervisor | Read/Write | Present |
|-------------------------------------|-------------|---------------------------|-------|----------|----------------|---------------|----------------|------------|---------|---------|
Entry Flags

• P: Whether the page or page table being pointed is loaded.
• U/S: 0-> no user access
• R/W: 0-> user read-only
• A : Accessed (set on swap-in)
• D : Dirty
  – (Only for page-table entry; you’ll use this at swap-out)
Setting Up Kernel Memory

- Allocate N_KERNEL_PTS (page tables)
- Fill them out until you’ve reach MAX_PHYSICAL_MEMORY
- PHYSICAL = VIRTUAL
- Need to be marked correctly
  - (especially SCREEN_ADDR)
Setting Up Process Memory

- Map kernel page tables starting at 0
- PROCESS_START (vaddr of code + data)
  - Use one page table and fill it out
  - It needs pcb->swap_size memory
- PROCESS_STACK (vaddr of stack top)
  - Use N_PROCESS_STACK_PAGES for the stack
Page Fault Handling

• Get a free page (from the page allocator)
• Swap into the page
• Set the page table entry to the page’s address and set present flag
• When do you need to flush the TLB?
Page Allocator

- If free page, return it
- Otherwise, you’ll need to swap a page out
- Some pages are pinned and you never evict them!
- In this project, implement any simple way (e.g., FIFO)
- Extra credit opportunity!
Things you’ll write

• memory.[ch]
• init_memory()
• setup_page_table(pcb_t *p)
• page_fault_handler()
• page_alloc(int pinned)
• page_replacement_policy()
• page_swap_in(int pagenumber)
• page_swap_out(int pagenumber)
• You will probably need to define structures to handle
What Can Make Your Life Easier

• One page table is enough for a process’ code and data memory space (starts at PROCESS_START)

• Some pages don’t need to be swapped out
  – *Need in this case means with respect to grading.*
Good Luck!

• Prepare for Design Reviews
• Enjoy your break?
Questions!