Recitation Week 11

(October 28th, 2014)

University at Buffalo
Fall 2014: CSE 421/521 Intro to Operating Systems

Sharath Chandrashekhara <sc296@buffalo.edu>

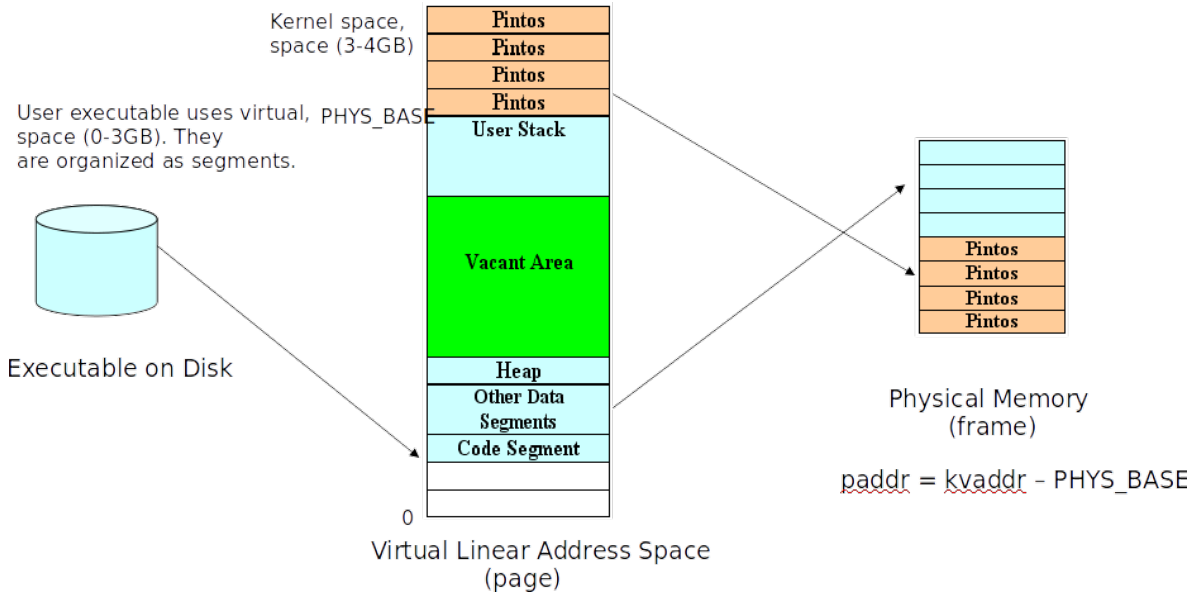
PintOS Project 3: Virtual Memory

- Project 2 of this class is really Project 3 of PintOS (manual)
- Project 3 builds on top of Project 2. But you do not have to implement Project 2
- A fully working code of Project 2 is provided to you
- However, PintOS still has some limitations:
  - The number and size of programs that can run is limited by the machine's main memory size.
  - In this project, you will remove that limitation

ReCap: Terminologies

- Virtual address
- Physical address
- Virtual Page
- Physical Page/Frame
- Page Table Entry
- TLB/Page Table
- TLB miss and Page Fault
- Supplementary Page Table
- Swap partition/slot
- Kernel address
- User address
- Frame table/Core map
- Cache algorithms

Virtual Memory in PintOS



What PintOS implements

- Basic virtual memory management
  - User processes live in virtual memory, cannot access the kernel directly
  - Kernel may access all memory
  - Functions to create and query page tables
- For swap in and swap out
  - Basic file system interface
  - You can read and write data to disk
  - You can read and write memory pages

How do you extend it?

- Implement a frame table
  - Keep track of Physical memory usage
- Implement dynamic allocation of pages
  - Lazy loading of pages
  - Allow the stack to grow dynamically
- Implement mmap() and munmap()
  - Create a table that keeps track of which files are mapped to which pages in each process

How do you extend it?

- Implement page swapping
  - If memory is full, take a page from physical memory and write it to disk
  - Keep track of which pages have been moved to disk
  - Reload pages from disk as necessary
  - Implement a swap table - Maps pages evicted from memory to blocks on disk
- Implement one of the cache replacement algorithms to choose which page to evict.

Code level changes

- Design the Page Table Entry (keep it as compact as possible)
- Design the Frame table (Core Map or Reverse Page Table). This shall be a global data structure
- Design the supplementary page table. This is a per-process data structure
- Enhance the *Page Fault Exception Handler* to load the correct address translation entries into the page table
- Implement the cache replacement algorithm to choose which page to evict. You can start with random and move to LRU clock implementation
- Feel free to write your own unit tests to test your components

General Tips - I

- Start early! You might have to write 500-700 lines of code for this project
- Start the project from reading the test code
- First make it work, if time permits, optimize
- Use static data structures like arrays whenever possible, even if it means wasting memory
- GDB is your friend. Spend time to learn it

General Tips - II

- User ASSERT generously. Debugging can get really hard on this project
- Panic the kernel when you don’t know what to do. Eg: Swap space is full
- Spend time to design your data structures: “Smart data structures and dumb code works a lot better than the other way around” - ESR
- Use git or any other source control tool you know (locally)
- Back up your code. Back up your code. Back up your code.

Next Recitation

- More implementation details and tips for this project
- Code run through
- Discussion of test cases
- Q & A

Credits

Slides were based on materials available online from similar courses in other Universities:

- PintOS manual
- CS140, Stanford University
- CS5600, Northeastern University
- CS3204, Virginia Tech.