Lab 3 More

Memory Management

Reminder

- Lab 3 Code **due Monday** 5/13/24
- Pset 5 Due Tomorrow! 5/10/24
- Pset 6 Out Tomorrow! 5/10/24
    - Due 5/17/24

Today’s Agenda

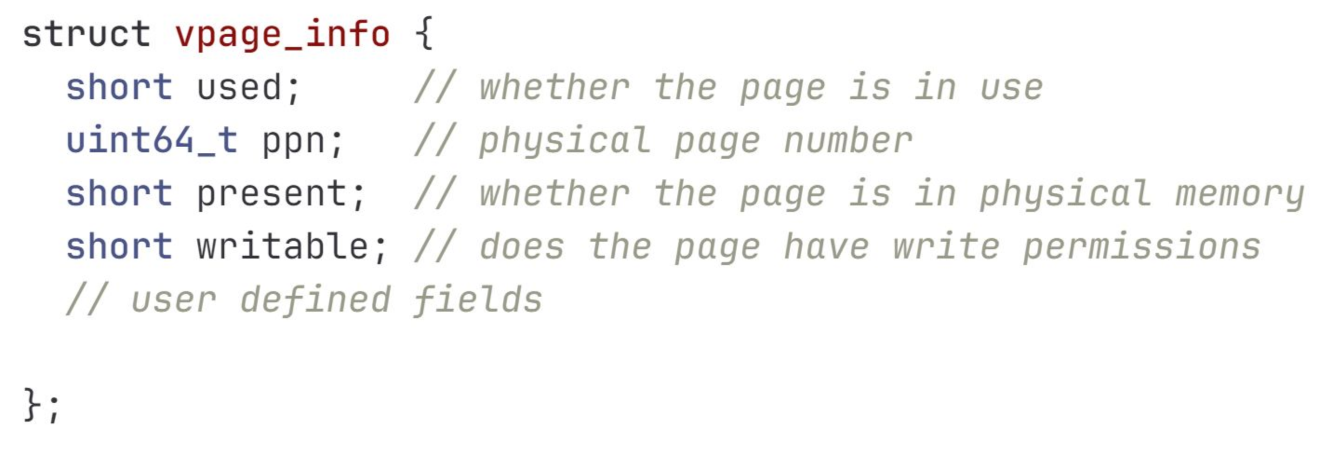
- More detail on vspace and vspace functions
- xk physical memory management
- Some discussion questions on lab 3
- Q&A time/Open OH

vspace Structs

Let’s talk virtual

Continuing from last week: you’ll be finagling and wrangling virtual memory in Lab 3. So let’s understand what you’re wrangling.

vpage_info

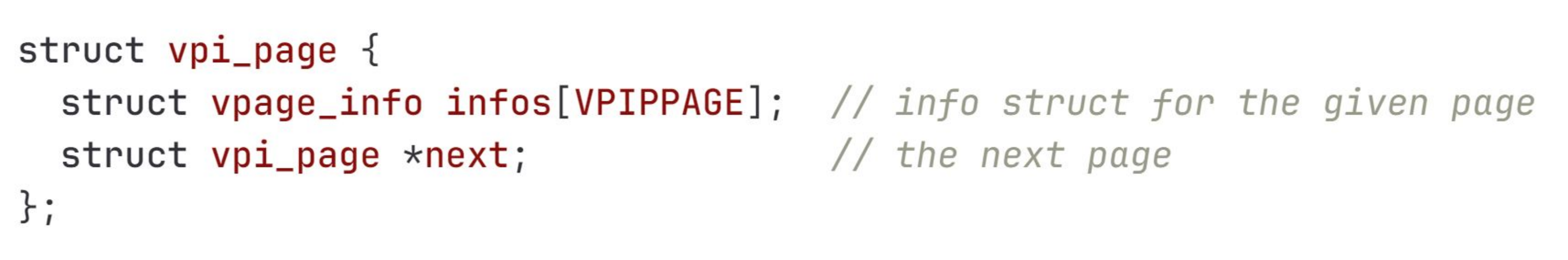


```
struct vpage_info {
  short used;      // whether the page is in use
  uint64_t ppn;    // physical page number
  short present;   // whether the page is in physical memory
  short writable; // does the page have write permissions
  // user defined fields

};
```

A struct vpage_info describes characteristics of the virtual page that we are pointing to, e.g used, physical page number, present, writable

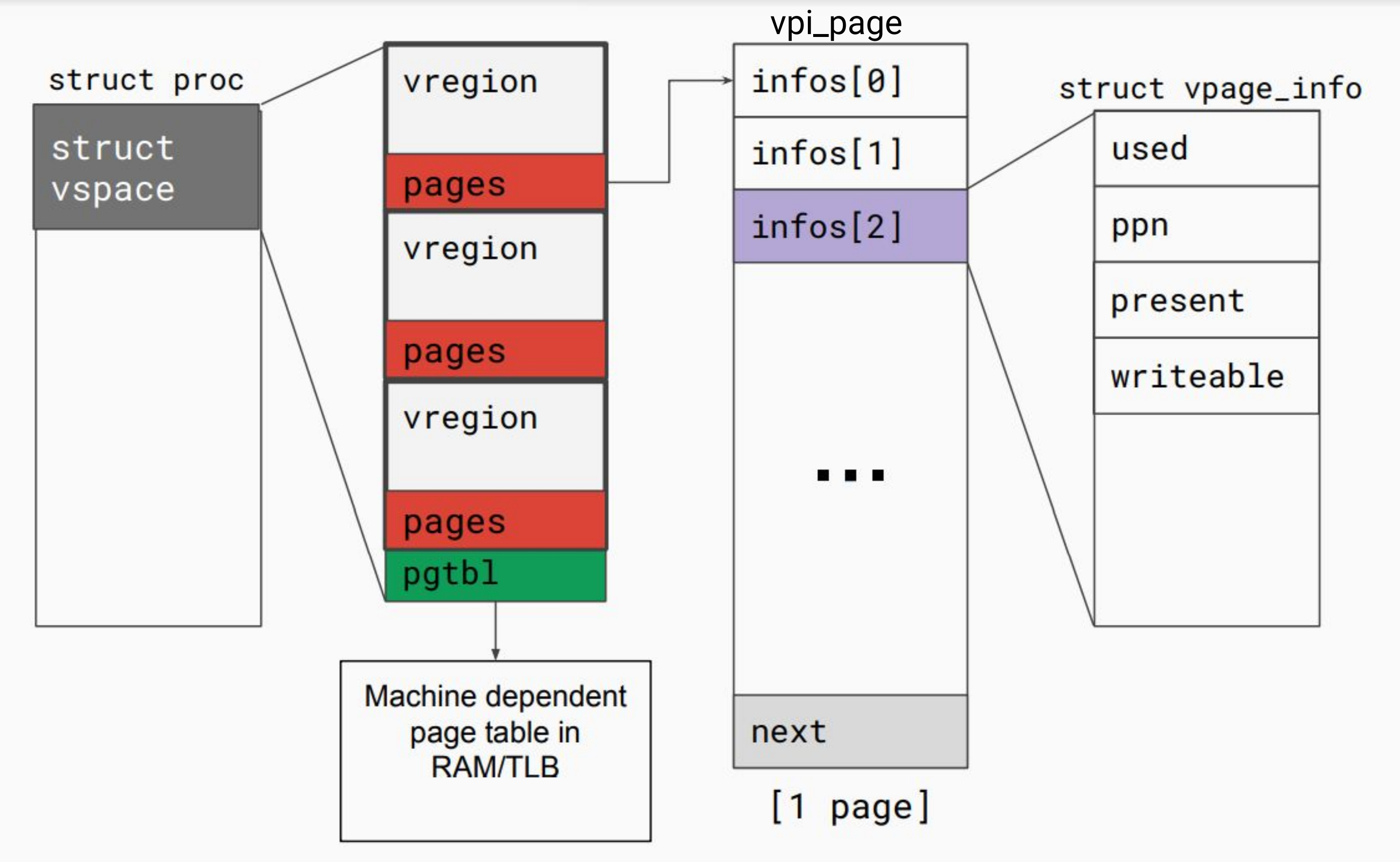
vpi_page



```
struct vpi_page {
  struct vpage_info infos[VPIPPAGE];  // info struct for the given page
  struct vpi_page *next;              // the next page
};
```

- A vpi_page is a container of vpage_info's
  - (vpi_page = “virtual page info page”).
- A vregion is made up of a linked list of vpi_pages.
  - (vregion can grow dynamically as needed)
- It stores an array of infos plus enough space for a pointer to a "next" vpi_page struct.

vspace Visual Diagram



vregions vs Page Tables

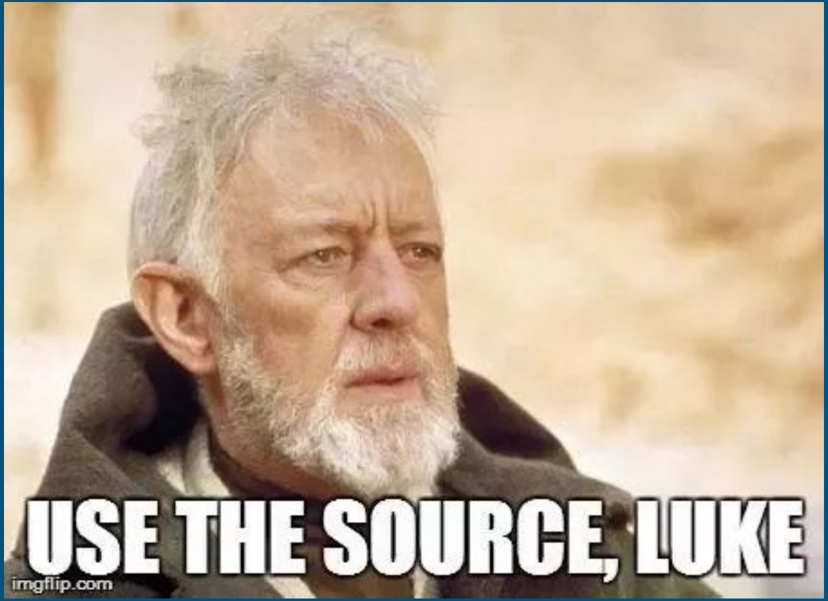
Ok so the vspace is made up of regions and the page table…

- What's the difference between xk’s vregions and the page table?

vregions vs Page Tables

- Can you make modifications to struct vpage_info?
- What happens if you make changes to vregions/vpage_info? Is it automatically reflected on the page table?

Time to practice!
How well do you know vspace.c?



Vspace Functions

For each question, there is a corresponding function in vspace.c

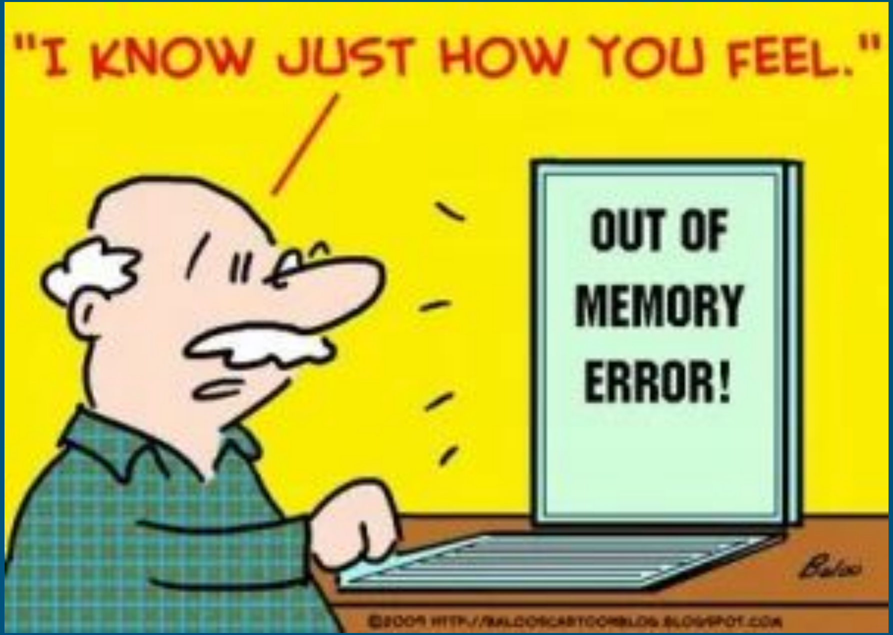
- Given a virtual address, how do you find which vregion it belongs to?
    - va2vregion
- Given a virtual address, how do you find its metadata (vpage_info)?
    - va2vpage
- How do you add a new virtual to physical mapping?
    - vregionaddmap
- How do you update the page table to reflect changes in vregion/vpage_info?
    - vspaceupdate
- How do you flush the TLB?
    - vspaceinstall

Vspace Events

- When would you want to flush the TLB?
  - When there's a change in page permission
- Do you need to flush the TLB after a new mapping is added?
  - No!

And that's the vspace side of things! But you'll need to deal with some physical frame bookkeeping too…

Physical Memory Management


"I KNOW JUST HOW YOU FEEL."
OUT OF MEMORY ERROR!
Baloo
©2009 HTTP://BALOOSCARTOONBLOG.BLOGSPOT.COM

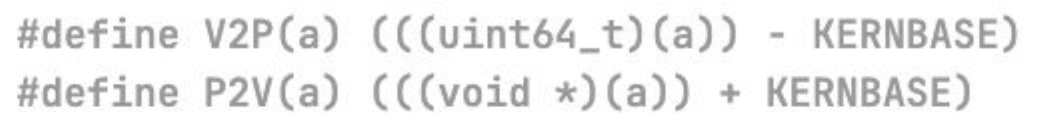
Motivation

- For COW fork you'll need to track refcounts on physical frames.
- Therefore: you'll need to interact with physical memory bookkeeping structures.
- Let's talk about that!

Add Oil
Direct translation of 加油 (jia you)
An exclamation expressing encouragement or support

Physical Memory Management

- Our QEMU instance emulates 16MB of physical memory
- It is entirely mapped into the kernel virtual address range starting at KERNBASE
- Can easily find the physical address backing a kernel virtual address: subtract va by KERNBASE
  - can the same thing be done on user virtual address?

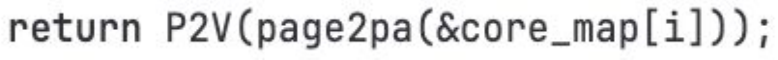


```
#define V2P(a) (((uint64_t)(a)) - KERNBASE)
#define P2V(a) (((void *)(a)) + KERNBASE)
```

Provided code has macros for doing physical/virtual conversions.

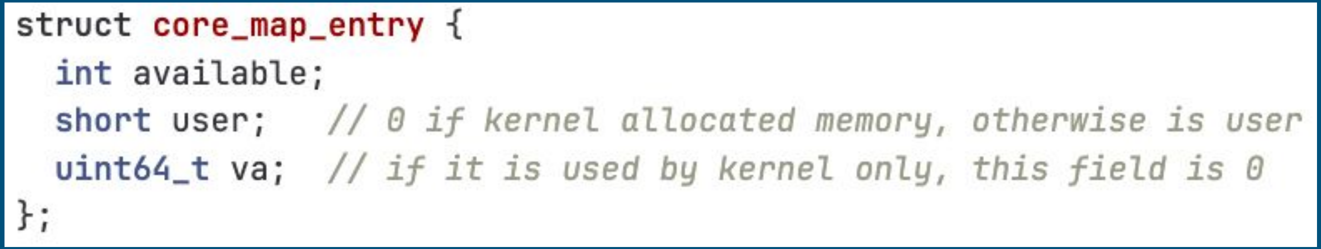
Physical Memory Allocation

- `kalloc` allocates a physical frame, it returns the kernel page mapped to the physical frame for ease of access



```
return P2V(page2pa(&core_map[i]));
```

- multiple system calls/kernel functions may call `kalloc` concurrently, what does `kalloc` do to keep these accesses safe?
- how does `kalloc` find a free frame?
  - by looking through metadata for frames (`core_map`)

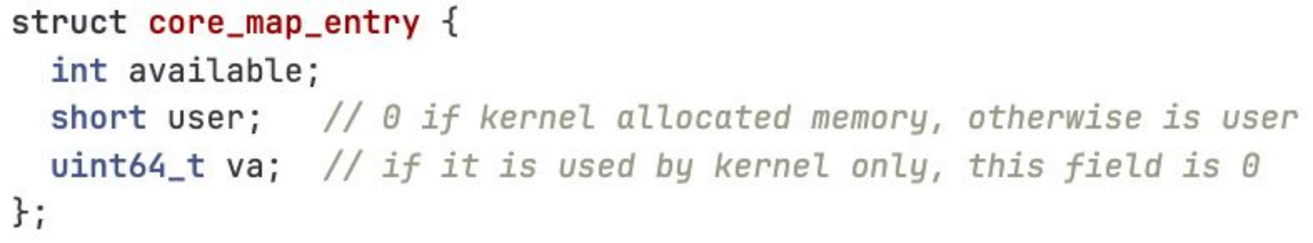


```
struct core_map_entry {
  int available;
  short user;    // 0 if kernel allocated memory, otherwise is user
  uint64_t va;   // if it is used by kernel only, this field is 0
};
```

Physical frame metadata

core_map_entry

- Access should be protected by the kmem.lock
- Can add to the struct to track additional information (refcounts)
  - Why do we care about refcount?
  - When will the refcount be greater than 1?



```
struct core_map_entry {
  int available;
  short user;   // 0 if kernel allocated memory, otherwise is user
  uint64_t va;  // if it is used by kernel only, this field is 0
};
```

physical frame metadata

kalloc and kfree Tips

You might want to update the physical frame ref counts in these functions…

- When we update ref counts, do we need to ensure synchronization?

When decrementing ref counts, make sure to always check if current ref count > 0!

- kfree is called on each frame during boot process. You can end up with -1 refcounts if you aren’t careful!

And that's the physical memory side of things! You are more than ready to tackle Lab 3 :)

Lab 3 FAQ

Error Codes FAQ

- Does the user bit (b2) configuration matter with regards to stack growth and COW cases?
    - No! Can happen in either kernel or user mode for both cases!
- When/where should I check error codes?
    - In trap()!

COW FAQ

- Do we need synchronization while modifying the **vspace** in page fault in COW fork?
    - Not needed -- current process has exclusive access to its own vspace (no multithreading)
    - **However, the <u>ref count</u> on the physical page could be concurrently modified**
- What can happen if a copy-on-write fork is not synchronized?
- What happens to a page that is already read-only before COW fork?

Helper Macros and Functions

P2V: translate physical addr to virtual addr

V2P: translate virtual address to physical address

PGNUM: translate physical address to page number

va2vpage_info: translate virtual address to vpi_info

Any questions?

Lab 3 Open OH