6.S081: Lab Q&A #2

Adam Belay <abelay@mit.edu>

Agenda

- Lab Q&A is an opportunity to better understand previous labs
  - Goal: Gain insights that help with future labs!
- Today's lab: COW
  - More difficult than previous labs (2-week assignment)
  - First lab with race conditions
- Some discussion of how Linux does MM

Why Copy-on-write (COW)?

- A common system-level optimization
- Critical with fork() -> exec() pattern
  - Prevents copying entire address space
  - Recall exec() discards address space
- More general: Key to deduplication
  - Use less memory by keeping a single copy of each unique page

Recap: Need VM and page faults

- VM plan
  - Mark PTE's as read only
  - Needed to avoid modifications to shared pages
- Page fault plan
  - Allocate new page for PTE
  - Copy old page contents to new page
  - Adjust PTE to enable writes

Recap: Page table entries (PTE)

63 54	53 28	27 19	18 10	9 8	7	6	5	4	3	2	1	0
Reserved	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
10	26	9	9	2	1	1	1	1	1	1	1	1

Figure 4.18: Sv39 page table entry.

Some important bits:

- **Physical page number (PPN)**: Identifies 44-bit physical page location; MMU replaces virtual bits with these physical bits
- **U**: If set, userspace can access this virtual address
- **W**: writeable, **R**: readable, **X**: executable
- **V**: If set, an entry for this virtual address exists
- **RSW**: Ignored by MMU

Recap: Gathering info for pgfault

1. The VA that caused the fault?
   - STVAL, or r_stval() in xv6
2. The type of violation that caused the fault?
   - Encoded in SCAUSE, or r_scause() in xv6
   - **12**: page fault caused by an **instruction** fetch
   - **13**: page fault caused by a **read**
   - **15**: page fault cause by a **write**
3. The IP and privilege mode where fault occurred?
   - **User IP**: tf->epc
   - **U/K**: SSTATUS, or r_sstatus() & SSTATUS_SPP in xv6

COW Lab: Key modifications

1. vm.c: uvmcopy()
   - Change PTE to read-only, mark COW using RSV bit
2. trap.c: usertrap()
   - Add logic to handle page faults
   - Add new method, cowpgflt() to handle COW faults
3. kalloc.c: throughout
   - Add support for reference counting
   - Add kget() to increment reference count
   - Change kfree() to decrement reference count
4. vm.c: copyout()
   - Call cowpgflt() to make sure we don’t write to a COW pg

COW solution walkthrough

Linux refcounting

- kref object manages refcount
- Refcount contained within an array of struct page

```
struct kref { refcount_t refcount; };

void kref_init(struct kref *kref)
void kref_get(struct kref *kref)
int kref_put(struct kref *kref, void (*release)(struct kref *kref))
```

Linux datastructures

- Vmarea list: describes virtual address layout
  - One per process
- Page array: describes physical pages
  - One per machine

Linux vmareas

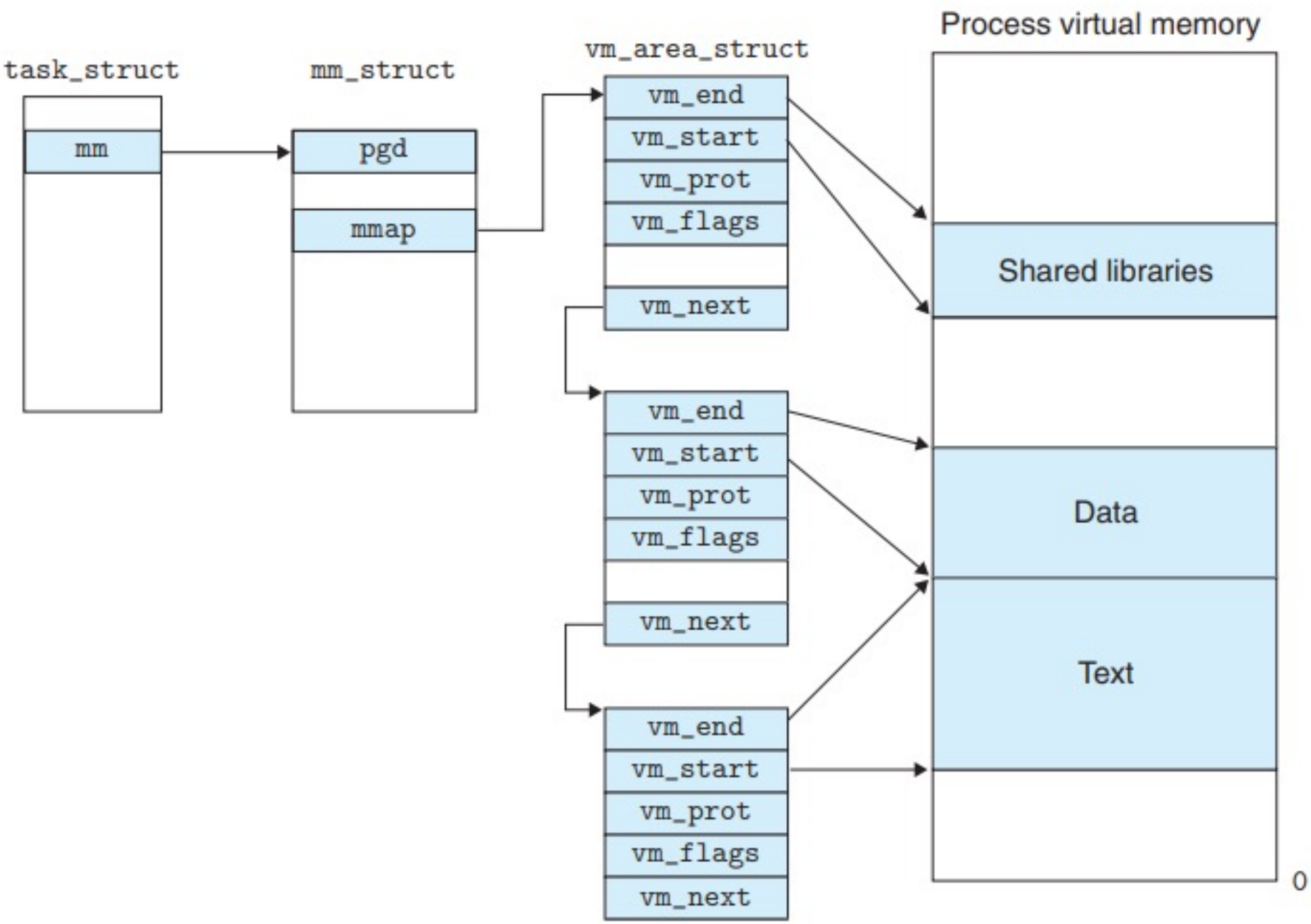

Figure 9.27 How Linux organizes virtual memory.

Linux pages

- Linux maintains a giant array of page structs, one for each page
  - Similar to COW solution
  - Each page has a refcount and has a lock
- Each page struct is several cachelines of metadata in practice