CS 202 - 001 : OS

Review Session 4

- [x] 0. Record
- [ ] 1. Background knowledge
- [ ] 2. Lab 4 Overview
- [ ] 3. Code walkthrough
- [ ] 4. Q & A

---

1. Background knowledge.

a, Virtual Memory :
- component of OS that gives process the illusion of having more memory than it actually has on the physical machine

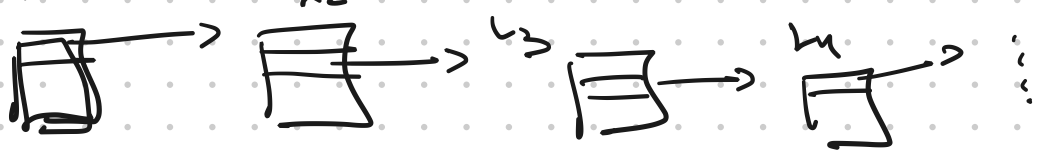
b, kernel : (~ OS)
- piece of the OS
- has full priviledge over the address space.

c, Page Table

9 bit for a page table.

4KB = 2^{12} B } → 2^{21} = 2MB.
2^9 / page table }

Q. How do we support 3MB VA on 2MB PA?



A: 2 L4 Page tables. Each Ptable = 2MB

---

2. Lab 4 Overview

Goal: - Virtual Memory Mgt.
- implement fork()

a, weensyOS DS.

- pageinfo: entry == physical page. Struct physical-pageinfo: owner & reference count.
- struct proc: process control blocks — pid: process id, state, registers, pagetable.

b, Helpful MACROS.

PAGESIZE = 4KB

PAGENUMBER =

{ bootloader.c.
x86-64.h

Where to look?

Kernel k-*.c/.h

Process p-*.c/.h

3 Helpful functions:

- vamapping virtual_memory_lookup (pagetable, va).

-> returns: vamapping {
pagenumber = -1
physical
}

- virtual_memory_map (k-hardware.c)

(pagetable, va, pa, size, perm, allocator)

-> Good to know how to use it.

- Assign_physical_page (physical address, owner)

-> ensure physical page is valid.
-> don't modify directly.

Step 1:

Clarification: kernel has its own ptable.

kernel.c

high level logic:

1. use vm_map():
2. Gives the right perm.

PTE_U : user accessible
PTE_P : present
PTE_W : writable
console : 0xB80000

Step 2: Give process its own page table

High level logic

- copy_pagetable (pagetable, owner)
- allocator func: gives back a new pagetable.
  - iterate through phys. memory
  - find a free phys. page
  - convert it to a pagetable.

- allocate a new ptable, look up ptable in the argument and map accordingly with VA → PA.

Step 3:

- Each process requests VA X, we use identity mapping
- Switch to find the first free physical pages & map accordingly.

Step 4

Motivation:

P1: use VA 600 → PA 1000

P2: use VA 600 → PA 2000

→ why can't P2 use VA 500?

(assuming VA 500 in P2's ptable not mapped)

→ Map stack page accordingly.

Step 5:

fork()?

- creates a child process
  → copies memory to child.

Parent's mem → Child's mem

Set return value

- How? -> registers (recall struct proc).

high level logic:

- Allocate a new ptable for child.
- virtual-memory-map - Use virtual_memory - lookups.
  VA -> PA.
- For writable, find a new phys. page & copy memory over (memcpy)

---

3. Code walkthrough

- virtual-memory-map
- process. setup
- exception: