CS 61 Lecture 12

Thursday, Oct 10th, 2019

Kernel Lecture #2

Allowed on midterm:

- internet to access course website, previous yrs of the website
- open book, open notes
- assembly manual

Patch

- a difference file

```
@@ -11,6 +11,11 @@ void process_main()
```

added 5 lines starting at line 6

```
@@ -137,8 +137,8 @@ void process_setup(...)
```

replaced some # of lines

versions

a) kernel/kernel.cc

b) kernel/kernel.cc

- apply a diff file w/ patch

```
patch < patch09-kernel-kernel.diff
```

Last Time

- infinite loop executed in the context of kernel by modifying its code

fabricate a pointer to memory

```
uint8_t* syscall = (uint8_t*) 0x40ad6;
syscall[0] = 0xeb;   // while (true)
```

syscall [1] = 0xff } infinite loop

[envoke-printf (0x0E00, "%d", sys-getpid());

infinite loop in x86-64

-6 (ff)

↳ signed 8 bit int = 0b 1111 1110 = -2

$-X = \sim X + 1$

flipping all bit +1

negative of ff = 0b0000 0010 = 2

program executes next line, -2 = current line

Where does the 1L go?

0x40ad6 → corresponds to syscall_entry function

What if Eve jumped to this syscall address rather than writing and invoking?

```
void (*f)() = (void(*)()) syscall;
f();
```

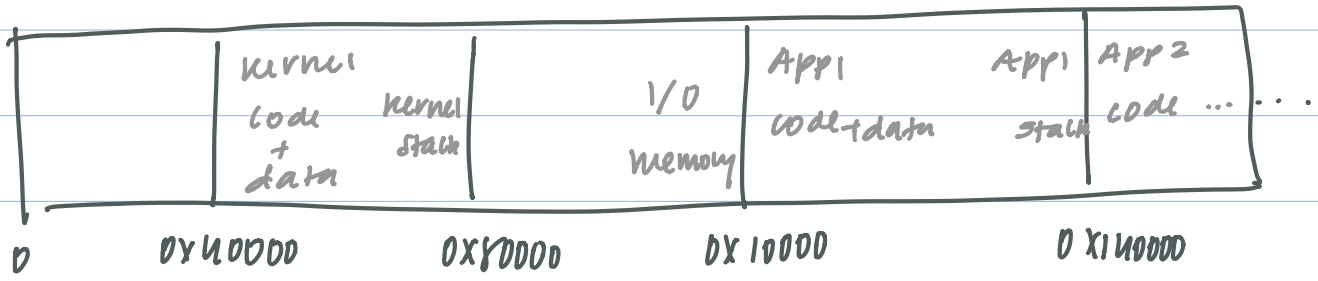
*code eve has compiled to: less obj/p-eve.asm

ff d0 callq *rax

↳ executing kernel code != changing privilege of which eve is running

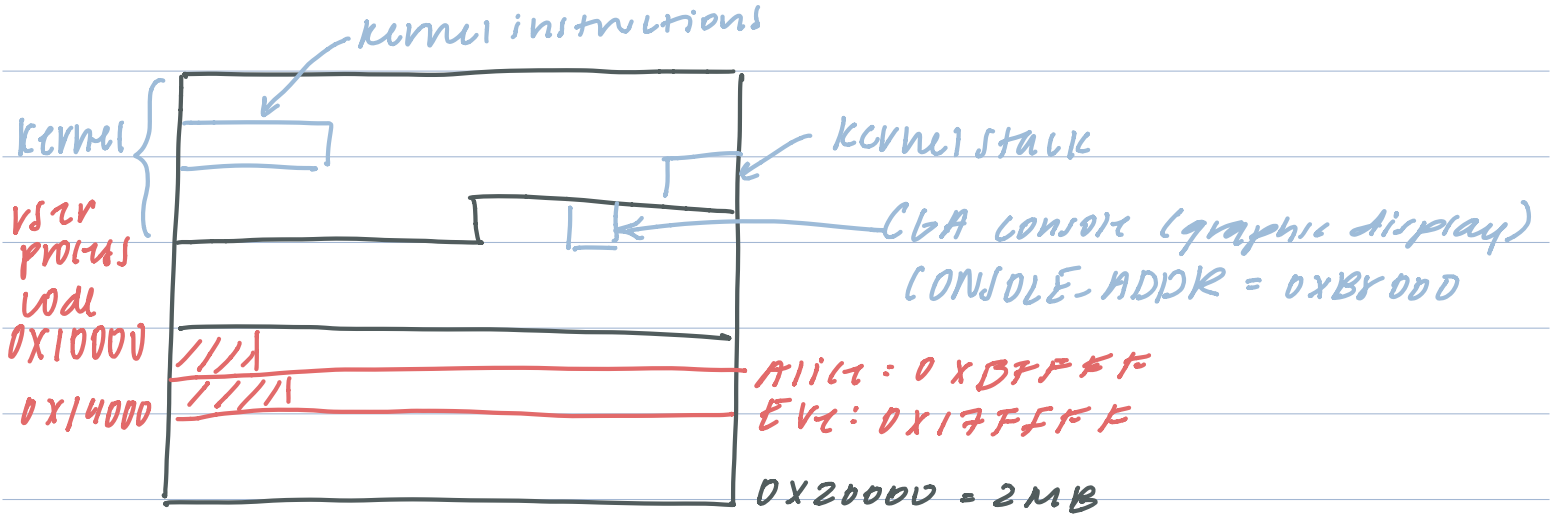
eve gains privilege when kernel gives it

X 86-64 Memory Layout



Weensy OS Memory Layout

physical memory



initializes kernel memory mappings for physical memory

```
for (vmiter it(kernel_pagetable, 0);
     it.va() < MEMSIZE_PHYSICAL;
     it += PAGESIZE) {
    if (it.va() != 0) {
        it.map(it.va(), PTE_P | PTE_W | PTE_U);
    } else {
        it.map(it.va(), 0);
    }
}
```

iterate over pages (2^{12} bytes)

maps to itself

allow anyone to access

cannot dereference null pointer

PTE_P: exists

PTE_W: writable

PTE_U: accessible to unprivileged code

add: if(it.va() >= PROC_START_ADDR) {
    it.map(it.val(), PTE_P | PTE_W, PTE_U);
}

remove PTE_U from previous

still causes exception 14 (page fault)
the is trying to access kbdc (on console page)

allow write/user access to console

change to:

if(it.va() >= PROC_START_ADDR ||
   it.va() == CONSOLE_ADDR) {
    it.map(it.val(), PTE_P | PTE_W, PTE_U);
}

Protected control transfer → SLOW.

processors mechanism for transferring control across privilege domains (less → more privileged)

X86-64

- kernel configures entry points for p.c.t.
- one entry point for every interrupt, fault, and trap
- one entry point for syscall instruction

interrupt / fault / trap

- kernel configures stack for protected control transfer at boot time
- when interrupt / fault / trap occurs:
  1. processor changes to configured stack
  2. processor pushes old stack pointer onto new stack + other critical regs (%rip, %cs)
     - %rip: value of instruction that caused fault
     - %cs: has privilege bits
  3. processor installs the kernel's configured entrypoint and %cs register ← changes privilege at entrypoint
     - entrypoint: changes instruction that is executed next
  4. kernel takes control (kentry.S)

iretq : undos work (return to process)

↳ values loaded from process descriptor

skip over faulting instruction

regs -> reg_rip += 8

↳ faulting instruction is 8 bytes long

kernel's configured entrypoint

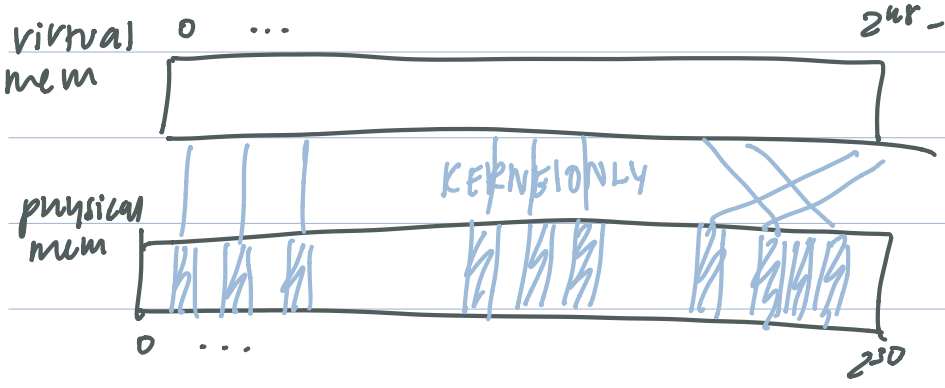
- jump to sys_call_entry upon initialization

kill the process

current -> state = P_BROKEN

Implementation of virtual memory

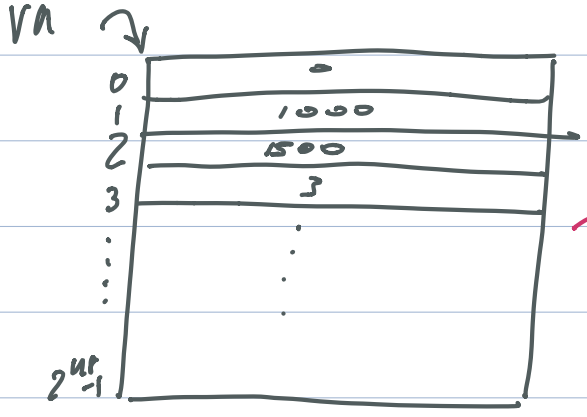
x86-64 page tables



kernel sets up

processor implements

Strawman: Byte map



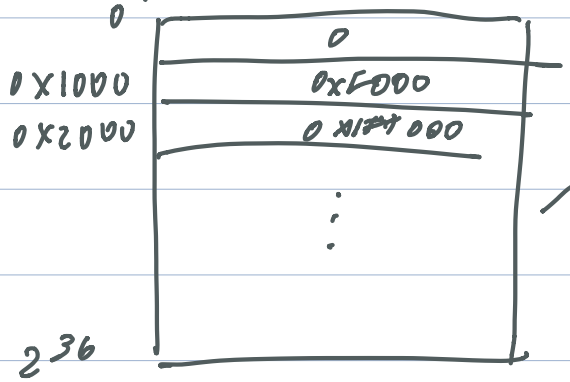
mapping for every byte

→ problem: 2^{51} bytes is huge.

pa = pt[va]

Strawman: Page Map

Map blocks (2^{12} bytes), aka "pages"



→ $2^{37} = 2^{36} \times 2^{3}$ bytes

4096

pa = pt[va / PAGE_SIZE] +

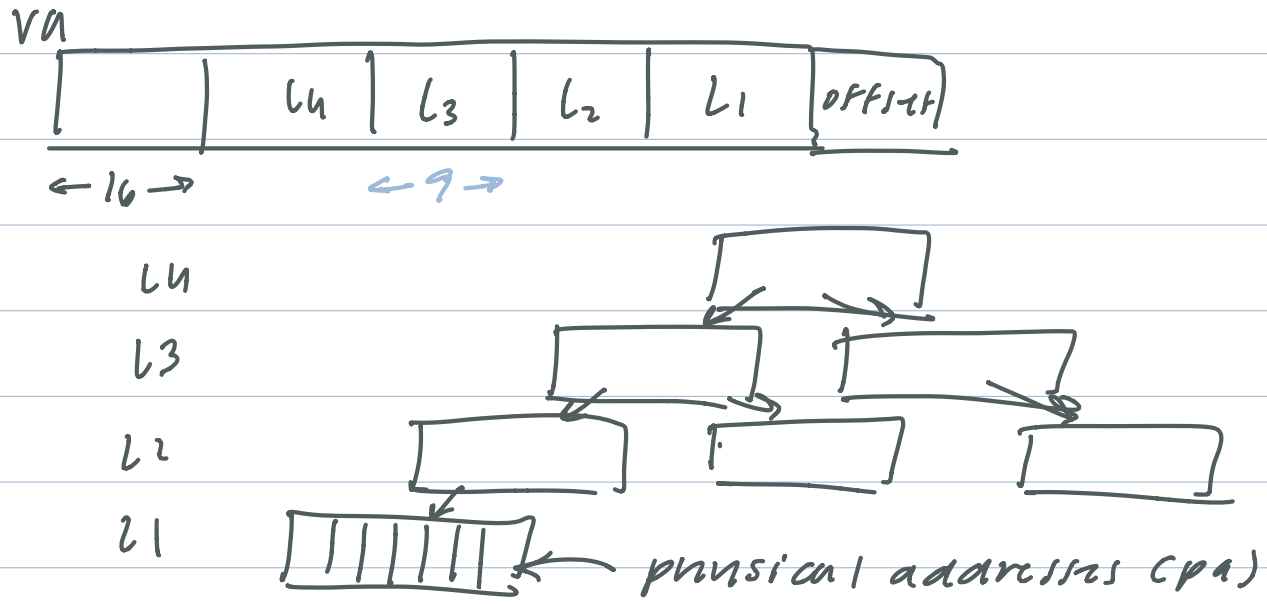
(va & (PAGE_SIZE - 1))

4095

lower 12 bits

True design (compress out 0s)

(compress address space: process will use 2^{16} bytes



2^{14} bytes in Memory (instead of 2^{39})