MKC - Exercise 1

Benjamin Engel

2014-07-02
Overview

- Kernel init/bootstrap
- Setup segments descriptor table
- Prepare initial page table, enable paging
- Jump to above 3GB, switch kernel stack
- Init serial port for debugging
- GDT, IDT, GSI, PIC

Hands-on
- Iret
- Sysenter/sysexit
- Very simple syscalls, e.g. add
Roadmap

- Brief intro/review on kernel bootstrapping

- Start within minimal kernel
- Leave kernel to userland via iret
- Reenter via sysenter
- Do very basic syscalls (nop, add, ...)

• Open src/start.S
• Hard-coded segment descriptor table
• Execution starts at symbol __start
• Setup boot page table
• Enable paging
• Load segment selectors
• Call init()
Initial Machine State

- Protected mode, no paging, but segmentation
- All segments: base 0, limit 0xFFFF FFFF
- CS: 32bit r+x code segment
- DS, ES, FS, GS, SS: 32bit r+w data segment (see Multiboot Specification for details)
Setup Memory

- Setup boot page table
Setup Memory

- Setup boot page table
- Enable paging, thus use page tables to fetch next instruction (therefore need 1:1 mapping)
• Setup boot page table
• Enable paging, thus use page tables to fetch next instruction (therefore need 1:1 mapping)
• Jump to high memory
  - This changes the EIP, but the 'old' stack from 1:1 mapping is still in use, need to switch later
- Open src/init.cc
- Setup serial port for early debug output
- Map new kernel stack
- Setup GDT, IDT, GSI, and TSS
- Init PIC, mask all IRQs or install handlers
- Prepare Sysenter (CS,EIP,ESP)
- Switch stack and unmap 1:1 mapping
• Open src/ec.cc: `root_invoke()`

• Create new thread, or EC (execution context)
  – Map 1 page user stack (at address 0x1000)
  – Map 1 page user code (at address 0x2000)

• Prepare stack frame to be used with `iret`
  – User code segment + instr. pointer (CS, EIP)
  – User stack segment + stack pointer (SS, ESP)
  – No Data segment for now

• IRET: loads CS:EIP, SS:ESP and EFLAGS
IRET stack layout

- (kernel) ESP points to array with CS:EIP, EFLAGS and SS:ESP
- IRET (atomically) loads registers and switches from privilege level 0 to 3
- Fetches and starts executing first instruction from new instruction pointer
• Open src/ec.cc, root_invoke()
• Map page with user code to address 0x2000

• User code starts in function usercode, thus
  ```
  mword code = reinterpret_cast<mword>(&usercode);
  ```

• Adjust new EIP to point within page at 0x2000
  ```
  code = (code & PAGE_MASK) + 2 * PAGE_SIZE;
  ```

• Handcraft stack frame and **IRET**
asm volatile (  
    "nop;"  
    : <out> : <in> : <clobber> );

mword i=2, j=3;  
asm volatile (  
    "add %ebx, %eax;"  
    : "+a" (i) : "b" (j) );
printf ("%d %d\n",i,j);

• Load esp with addr of stack frame and do 'iret'
• Prepare array with 5 elements and `iret`
  - `Code`: user instruction pointer to exit to
  - `SEL_USER_CODE`: new CS (include/selectors.h)
  - `0x202`: EFLAGS, just set interrupt enabled flag
  - `2 * PAGE_SIZE`: new stack pointer
  - `SEL_USER_DATA`: new SS stack segment
• Open `src/usercode.cc`: `usercode()`
  - `1st` Fault immediately
  - `2nd` reenter kernel via `sysenter`
  - `3rd` prepare `sysexit` by loading `ecx` and `edx` with proper values (ESP and EIP `after` returning)
  - `4th` do simple system calls, like add 2 numbers
- Open src/usercode.cc, function usercode()

- To check if everything is ok, fault immediately
  - asm ("ud2"); → exception #6
  - Ec::handle_exc 0x6 (eip=0x2016 cr2=0x0)

- Force a page fault by reading or writing to an address somewhere below 0x1000
  - Ec::handle_exc 0xe (eip=0x2016 cr2=0x23)