MKC – Exercise 1

Nils Asmussen and Viktor Reusch

2024-05-02
Roadmap

- Brief intro/review on kernel bootstrapping
- Start within minimal kernel
- Leave kernel to userland via `iret`
- Reenter via `sysenter`
- Do very basic system calls (`nop`, `add`, ...)
• 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
• Exact values are undefined
• See Multiboot specification for details
$ git clone https://os.inf.tu-dresden.de/repo/git/mkc.git
$ git checkout exercise1

# build it
$ make

# run it
$ make run
• 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()`
Setup Memory

- Setup boot page table
• 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 Memory

- 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
Init() ...ialization

- 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)`
- Switches stack

![Diagram showing physical memory, virtual memory, and kernel stack with EIP marker.](image)
Init() ...ialization

- Again, *src/init.cc*, `bootstrap()`
- Removes 1:1 mapping
- Flushes *TLB*
- Creates new EC (thread) for our user-code
- Switches to that EC
Getting out of the kernel ...

- Open `src/ec.cc, root_invoke()`

- Prepare address space
  - 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 + instruction 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

<table>
<thead>
<tr>
<th>10</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>ESP</td>
</tr>
<tr>
<td>4</td>
<td>EFLAGS</td>
</tr>
<tr>
<td>0</td>
<td>CS</td>
</tr>
<tr>
<td></td>
<td>EIP</td>
</tr>
</tbody>
</table>

- (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
Getting out of the kernel ...

- User code starts in function usercode, thus
  
  \[
  \text{mword code} = \\
  \text{reinterpret_cast<mword> (&usercode)};
  \]

- Adjust new EIP to point within page at 0x2000
  
  \[
  \text{code} = (\text{code} \& \text{PAGE_MASK}) + 2 \times \text{PAGE_SIZE};
  \]

- Handcraft stack frame and `iret`
asm volatile(
    "nop;"
    : <out> : <in> : <clobber>
);

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

• **Load ESP** with address 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`)
  - `0x200: EFLAGS`, just set interrupt enabled flag
  - `2 * PAGE_SIZE`: new stack pointer
  - `SEL_USER_DATA`: new SS stack segment

• Open `src/usercode.cc::usercode()`
  - 1\textsuperscript{st} fault immediately
  - 2\textsuperscript{nd} reenter kernel via `sysenter`
  - 3\textsuperscript{rd} prepare `sysexit` by loading `ECX` and `EDX` with proper values (`ESP` and `EIP` after returning)
  - 4\textsuperscript{th} 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)`