xv6©-rev10

(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)

Entering the Kernel, main

Carmi Merimovich

Tel-Aviv Academic College

September 18, 2017

Where are we?

1. POWERUP.
   - The Boot processor executes instructions. MMU inactive.
2. ROM code loades 512-bytes boot block to address 0x07C0 and up.
   - ROM terminates by JMPing to address 0x07C0.
3. Boot block code loads the kernel from ide0 into 0x00100000 and up.
   - Boot block code terminates by JMPing into the kernel's entry point.
   - The entry point address is found at a fixed location in the kernel ELF.
   - The entry point code is in Assembly and is labeled `entry`.
4. `entry` sets up a temporary kernel programming model.
   - MMU is activated with a table coding the following translation:

$$[\texttt{0x80000000}, \texttt{0x803FFFFF}] \mapsto [\texttt{0x00000000}, \texttt{0x003FFFFF}]$$

   - `esp` is set to the end of a 4KB buffer.
   - `entry` finishes by JMPing into `main`.

main

Since we got here from **entry**, this code is run by the Boot processor.

- **main** calls routines initializing the different subsystems.
- We study each initialization together with its subsystem.
- Most subsystems need one initialization per (computer) system.
- Initialization which are needed per-cpu are done in **mpmain**.

The application processors begins at **entryother**, then proceed to **mpenter**.

main

```
1216 int main(void) {
       kinit1(end,
              P2V(4*1024*1024));
       kvmalloc();
       mpinit();
       lapicinit();
       seginit();
       picinit();
       ioapicinit();
       consoleinit();
       uartinit();
       pinit();
       tvinit();
       binit();
       fileinit();
       iinit();
       ideinit();
       if(!ismp)timerinit();
       startothers();
       kinit2(P2V(4*1024*1024),
              P2V(PHYSTOP));
       userinit();
       mpmain();
     }
```

```
1251 static void mpmain(void) {
       idtinit();
       xchg(&(mycpu()->started),
       scheduler();
     }
```

Starting auxiliary processors

In **startothers()**, for each application processor the boot processor executes the following:

1. Copies the **entryother** code into address 0x80007000.
   - **entryother** is a replacement of the **entry** routine.
2. Through the lapic instructs the AP to start executing at 0x0700.
3. Waits for the AP be done with initialization.

Application processor initialization

1. `entryother` sets up a temporary kernel programming model.
   - MMU is activated with a table coding the following translation:

     $$[\texttt{0x80000000}, \texttt{0x803FFFFF}] \mapsto [\texttt{0x00000000}, \texttt{0x003FFFFF}]$$

   - `esp` is set to the end of a 4KB buffer.
   - `entryother` finishes by JMPing into `mpenter`.

```
1240 static void mpenter(void) {
       switchkvm();
       seginit();
       lapicinit();
       mpmain();
     }
```

The initializations we study now

```
kinit1(end, P2V(4*1024*1024)); // phys page alloca
kvmalloc();                    // kernel page table
   ⋮
seginit();                     // set up segments
   ⋮
pinit();                       // process table
   ⋮
   ⋮
kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must co
userinit();                    // first user process
mpmain();
```