TODO: write a nice title here

Thomas Hybel

Aarhus University
October 2017

Abstract

TODO: write an abstract here.

Contents

Introduction

- introduction - what are we trying to do - how are we doing it - following the course from MIT - why are we doing it - overview of the coming chapters

Lab 1

In this lab we wrote initialization code for our operating system. This code sets up a rudimentary page table and switches the processor from 16-bit real mode to 32-bit protected mode. We also wrote a boot loader, which is a small program that loads the main kernel from disk and transfers control to it.

The boot process

To understand the purpose of a boot loader, it is first necessary to have an overview of the process which an x86 machine goes through upon startup.

When an x86 machine starts, its BIOS code is run. The BIOS initializes some of the system's hardware components (e.g., keyboard, graphics card, and hard drive). The BIOS will then load one sector (512 bytes) from the boot medium into memory at a hard-coded address (0x7C00). Once this first sector is loaded into memory, the BIOS will transfer execution to the loaded code.

The first sector will typically contain a small program known as the boot loader. Its purpose is to load the main kernel from disk and transfer execution to it.

Before the boot loader loads the kernel, we first run some initialization code which sets up a more comfortable environment for the boot loader and kernel to work in.

Switching processor mode

When the BIOS jumps into our code, the processor is running in 16-bit real mode. However the code produced by a modern compiler expects to run in 32-bit protected mode. We therefore needed to write code which performs this switching of processor modes.

The main difference between real mode and protected mode is how address resolution is performed. In real mode, accessing a physical address is done using a segment selector register and a general-purpose register. In contract, protected mode has the concept of a page table, which maps between virtual and physical addresses. Since we want our operating system to use virtual memory, we certainly want to switch to protected mode.

To switch to protected mode, a bit needs to be set in the status register `cr0`. We do so with the following assembly code:

```
# protected mode enable flag
.set CR0_PE_ON, 0x1
movl    %cr0, %eax
```

```
orl     $CR0_PE_ON, %eax
movl    %eax, %cr0
```

To

- switching from real to protected mode - switching from 16-bit to 32-bit mode

- lab 2 - alloc and free physical pages - page table management

- lab 3 - goal: make user-space process run - functions for keeping track of proceses - ELF loader - context switching - handling exceptions - handling syscalls

- lab 4 - preemptive multitasking - starting multiple processors - simple scheduler with yields - preemption (clock signals) - process forking with CoW - simple fork - CoW fork - IPC

- lab 5 - file system - shell

- lab 6 - networking: writing an e1000 card driver

- graphics

- hardware

- conclusion

References