HOW TO MAKE AN OPERATING SYSTEM FROM SCRATCH

Samy Pessé

Published with GitBook

Table of Contents

How to Make a Computer Operating System

Online book about how to write a computer operating system in C/C++ from scratch.

Caution: This repository is a remake of my old course. It was written several years ago as one of my first projects when I was in High School, I'm still refactoring some parts. The original course was in French and I'm not an English native. I'm going to continue and improve this course in my free-time.

Book: An online version is available at http://samypesse.gitbooks.io/how-to-create-an-operating-system/ (PDF, Mobi and ePub). It was generated using GitBook.

Source Code: All the system source code will be stored in the src directory. Each step will contain links to the different related files.

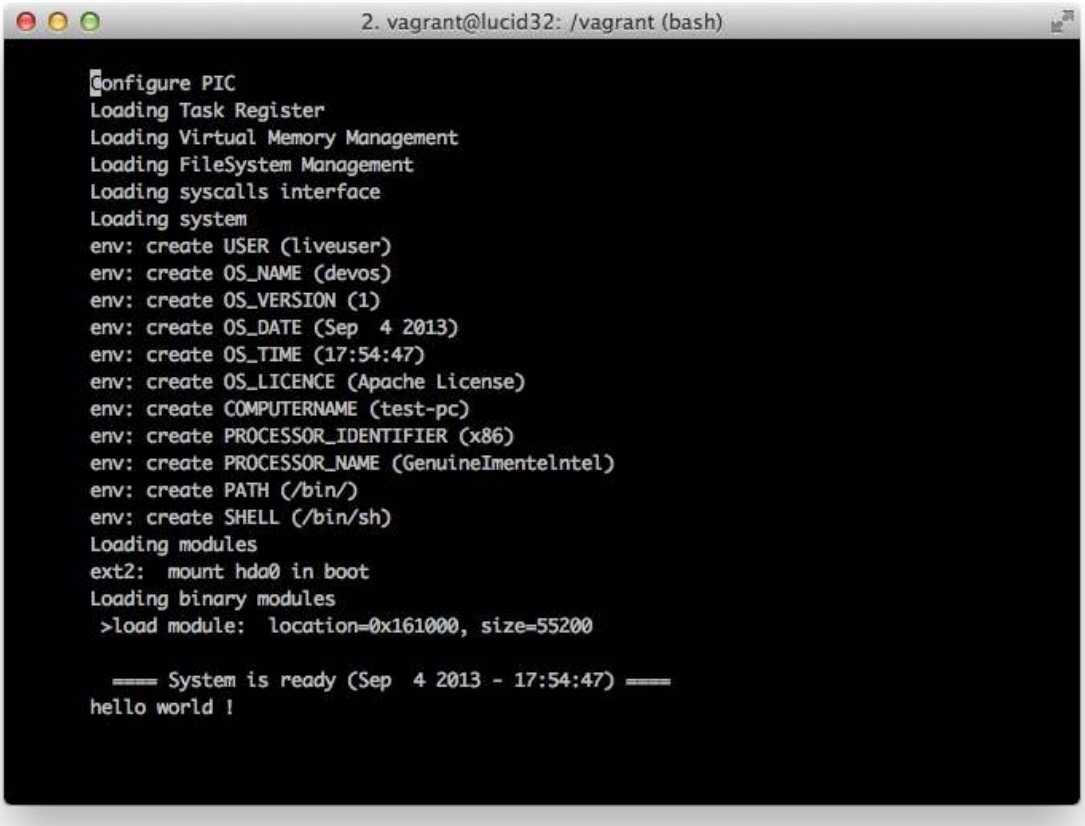
Contributions: This course is open to contributions, feel free to signal errors with issues or directly correct the errors with pull-requests.

Questions: Feel free to ask any questions by adding issues or commenting sections.

You can follow me on Twitter @SamyPesse or GitHub.

What kind of OS are we building?

The goal is to build a very simple UNIX-based operating system in C++, not just a "proof-of-concept". The OS should be able to boot, start a userland shell, and be extensible.


2. vagrant@lucid32: /vagrant (bash)
Configure PIC
Loading Task Register
Loading Virtual Memory Management
Loading FileSystem Management
Loading syscalls interface
Loading system
env: create USER (liveuser)
env: create OS_NAME (devos)
env: create OS_VERSION (1)
env: create OS_DATE (Sep 4 2013)
env: create OS_TIME (17:54:47)
env: create OS_LICENCE (Apache License)
env: create COMPUTERNAME (test-pc)
env: create PROCESSOR_IDENTIFIER (x86)
env: create PROCESSOR_NAME (GenuineImentelntel)
env: create PATH (/bin/)
env: create SHELL (/bin/sh)
Loading modules
ext2: mount hda0 in boot
Loading binary modules
>load module: location=0x161000, size=55200
==== System is ready (Sep 4 2013 - 17:54:47) ====
hello world !

Chapter 1: Introduction to the x86 architecture and about our OS

What is the x86 architecture?

> The term x86 denotes a family of backward compatible instruction set architectures based on the Intel 8086 CPU.

The x86 architecture is the most common instruction set architecture since its introduction in 1981 for the IBM PC. A large amount of software, including operating systems (OS's) such as DOS, Windows, Linux, BSD, Solaris and Mac OS X, function with x86-based hardware.

In this course we are not going to design an operating system for the x86-64 architecture but for x86-32, thanks to backward compatibility, our OS will be compatible with our newer PCs (but take caution if you want to test it on your real machine).

Our Operating System

The goal is to build a very simple UNIX-based operating system in C++, but the goal is not to just build a "proof-of-concept". The OS should be able to boot, start a userland shell and be extensible.

The OS will be built for the x86 architecture, running on 32 bits, and compatible with IBM PCs.

Specifications:

- Code in C++
- x86, 32 bit architecture
- Boot with Grub
- Kind of modular system for drivers
- Kind of UNIX style
- Multitasking
- ELF executable in userland
- Modules (accessible in userland using /dev/...) :
  - IDE disks
  - DOS partitions
  - Clock
  - EXT2 (read only)
  - Boch VBE
- Userland :