

# *Creating an operating system*

by

Tomasz Łaśko

[tomasz@lasko.pl](mailto:tomasz@lasko.pl)

[www.tomasz.lasko.pl](http://www.tomasz.lasko.pl)

# *Contents*

- ◆ Introduction
- ◆ Examples of "home made" operating systems
- ◆ Protected mode
- ◆ Tools and environment for developing OS's
  - ◆ Bootloaders
  - ◆ Virtual Machines
  - ◆ Compilers
- ◆ Resources

# *Introduction*

- ◆ Everybody wants to do something really good. For a computer engineer it can be an operating system
- ◆ An ambiguous, demanding task with a wide variety of different topics and problems
- ◆ Linux also started this way. Maybe in near future your OS will be known and used worldwide too :)
- ◆ *„If you want to travel around the world and be invited to speak at a lot of different places, just write a Unix operating system.“* – Linus Torvalds – creator of the Linux operating system

## *Examples of "home made" and other exotic operating systems*

- ◆ Visopsys – Visual Operating System  
[www.visopsys.org](http://www.visopsys.org)
- ◆ O3ONE – The Object Oriented OS  
[www.o3one.org](http://www.o3one.org)
- ◆ Syllable [www.syllable.org](http://www.syllable.org)  
*„The goal of Syllable is to create a reliable and easy-to-use open source operating system for the home and small office user” - Syllable OS creators*
- ◆ There is a whole bunch of many other OS'es like these above

# *Protected mode (PM)*

- ◆ In x86 family PM appeared for the first time in 80286, but it was fully featured in 80386

Why this strange name? And what it does?

- ◆ *Protects* OS kernel from user programs
- ◆ *Protects* good programs' code and data from errorous programs
- ◆ *Protects* CPU IO address space (direct access to PC hardware) from user programs. This address space should be only visible to the kernel and hardware drivers software
- ◆ Allows to *virtualize* whole memory address space and makes it very flexible

# *Running code under protected mode*

- ◆ At the time of booting, an x86 CPU is in 16bit real mode (for 8086 backward compatibility) and the only available software „library” (or you can call it API) is BIOS. It is used i.e. by DOS (a real mode OS) to access the PC hardware
- ◆ Most pieces of BIOS code are also 16bit and don't support protected mode so you can't use BIOS calls in PM like DOS does, and even when you can, this is very very very inconvenient
- ◆ If you can't use BIOS then there is nothing else you can use – you have to write *whole* software on your own
- ◆ But you can (and in most cases should) access all the areas of a PC system. *You* rule here now :)

# Bootloaders

- ◆ There are many bootloaders that can boot any OS, but then the OS must have its own bootloader to initialize the hardware, enter protected mode and load the rest of the system – all in this system's specific way
- ◆ There are some popular bootloaders which can do some of this thing for us
- ◆ Most popular (especially in the group of OS developers) and one of the most advanced bootloaders is **GRUB**.
- ◆ Apart from being a new alternative for LILO in booting Linux OS, GRUB gives us very advanced tools, for example a shell like console which runs in time of PC boot, in which you can manually enter all GRUB's commands
- ◆ GRUB also initializes protected mode for us
- ◆ But in most cases you will have to or just want to create your own OS boot-and-initialize code, which will behave just like *you* want to

# *Virtual Machine*

- ◆ Here is a quite ridiculous method of developing an OS: Compiling your operating system, dumping it on a boot disk and then rebooting your computer, booting up your developed system – all that activity just to test some little change in the code – and to do it over and over again, every time
- ◆ Another problem is met during tests of your system when you want to debug your system – it is a hard task when your OS runs on some PC and you can't be sure what is going inside
- ◆ Virtual machine is a tool, which helps us solve these and many other problems. It's a complicated program which emulates a real computer, creates a virtual environment similar to a real computer system, inside which it runs any operating system you want. And that operating system cannot even know that it is run inside a virtual machine
- ◆ A system which runs inside a Virtual Machine is called the guest system
- ◆ Your own real system which manages and runs all virtual machines is called the host system

# *Virtual Machine features*

A typical virtual machine supports the following features:

- ◆ Virtual hard disk images stored in files. Floppy and CD-ROM images support.
- ◆ „Hibernating” the guest system at *any* time and resuming it later
- ◆ Networking between a virtual machine and:
  - ◆ other virtual machines
  - ◆ your host OS
  - ◆ other real, physical stations on the network that your computer is connected to
- ◆ Sound (usually by emulating a Sound Blaster 16 card)
- ◆ Connecting to a virtual machine's virtual communications port (COM, LPT). For example your own program which is running on your real (host) OS can communicate with the virtual machine (i.e. by a named pipe) and the virtual machine thinks it is some external device attached to the communications port

# *Most known Virtual Machines*

- ◆ **VMware Workstation** [www.vmware.com](http://www.vmware.com)
  - ◆ Very good networking
  - ◆ Good support for all 32-bit MS Windows and Linux platforms
  - ◆ Free 30-day trial available
- ◆ **Bochs** (pronounce: *box*) <http://bochs.sourceforge.net>
  - ◆ Open source project
  - ◆ Best for running and debugging your own OS
  - ◆ Includes (as an option selected during compilation of Bochs) a special debugging user interface, which allows you to take control of every byte of memory and every register of CPU inside the virtual machine at any time
  - ◆ Some interesting supported features:
    - ◆ Networking
    - ◆ VMware 3 virtual disk support

## *Other known VM's*

- ◆ **Plex86** <http://plex86.sourceforge.net/>
  - ◆ Lightweight VM specialized to run only Linux x86 systems as a guest OS
- ◆ **Virtutech Simics**  
<http://www.virtutech.com/products/simics.html>
  - ◆ An enhanced VM platform
  - ◆ Free academic license for students
  - ◆ Currently supports Alpha, ARM, IA-64, MIPS, PowerPC, SPARC V9, x86, and AMD64 architectures

# *Compilers, linkers etc.*

- ◆ I definitely advise you to use GNU compilers and linkers. Most of them can be found at [www.gnu.org](http://www.gnu.org)
  - ◆ They give you many possibilities and flexibility
  - ◆ They also support variety of binary formats and platforms and I don't know if you can count them all
  - ◆ They are used by most Open Source OS developers and are used in most of examples and tutorials
  - ◆ And they are free :)
- 
- ◆ They're not ideal – for instance GCC supports only the flat memory model and if you want to manipulate segment descriptor registers and operate on different memory segments you have to use assembly
  - ◆ But still they are the best choice

# *GNU compilers and linkers*

- ◆ **GCC** – C and C++ compiler
  - ◆ A base tool of all Linux systems
  - ◆ **DJGPP** [www.delorie.com/djgpp/](http://www.delorie.com/djgpp/)
    - ◆ A special version of GCC to run under Microsoft systems (from DOS to Windows) prepared by DJ Delorie
- ◆ **LD** – a GNU linker
  - ◆ Able to link to binary files with layout defined by user in a file called linker script
- ◆ **AS** – GNU assembler
  - ◆ Used internally by GCC
  - ◆ Uses a strange AT&T language syntax
- ◆ **NASM** – The Netwide Assembler
  - ◆ <http://nasm.sourceforge.net/>
  - ◆ Very nice and easy language syntax

# *Internet resources*

Internet is full of OS-development resources. Here are some examples:

Documentation, specification, assembly:

- ◆ Dr Dobb's websites [www.ddj.com](http://www.ddj.com), [www.x86.org](http://www.x86.org)
- ◆ Hardware manufacturers, for example [www.intel.com](http://www.intel.com)
- ◆ Webster assembly site by R. Hyde <http://webster.cs.ucr.edu/>
- ◆ OSDev [www.osdev.org](http://www.osdev.org)

A review of this kind of OS'es:

<http://tunes.org/Review/OSes.html>

Other websites:

- ◆ SigOPS [http://www.acm.uiuc.edu/sigops/roll\\_your\\_own/](http://www.acm.uiuc.edu/sigops/roll_your_own/)
- ◆ OS-FAQ <http://www.mega-tokyo.com/osfaq2/>
- ◆ Christopher Giese (Geezer):  
<http://my.execpc.com/CE/AC/geezer/os/>

## *Books and other resources*

- ◆ Daniel W. Lewis, “Fundamentals of Embedded Software: Where C and Assembly Meet”, Prentice-Hall, 2001.  
<http://www.cse.scu.edu/~dlewis/>
- ◆ Linux kernel source code [www.kernel.org](http://www.kernel.org)  
An excellent source :) of practical knowledge
- ◆ Other Open Source OS'es source codes  
i.e. „home made” operating systems mentioned in  
this presentation

*The end*

Thank you!

If you have any questions or comments  
please feel free to mail me:

[tomasz@lasko.pl](mailto:tomasz@lasko.pl)