Steve Tranby  
Created: 9/29/2007  
Last-Modified: 9/29/2007

**README**

1. What is an operating system (OS)?
2. Why would I want to create my own OS?
3. What programming language should I use?
4. <http://www.nondot.org/sabre/os/files/Misc/os-faq/index.html>
5. Information
   1. Necessary to understand the intricacies of an operating system
   2. Compilers, Linkers, File Systems, Executable Formats
6. My Development System and Tools
   1. Virtual Machine
      1. Why?
      2. VMWare Server
      3. QEmu
      4. Bochs
      5. VirtualBox
   2. Assembler
      1. nasm
      2. masm
   3. C / C++ Compiler
      1. gcc and g++
      2. djgpp
   4. Linker
   5. SVN
7. Boot Loader
8. Kernel
9. Memory
   1. Virtual Memory
   2. Segmentation
   3. GDT / LDT
10. Interrupts and Exceptions
    1. Interrupt Service Routines (ISRs)
    2. Interrupt Requests (IRQs)
    3. Programmable Interrupt Controller (PIC)
11. Managing Resources
    1. Manage Memory
       1. LRU
       2. Swapping
    2. Manage Processes
    3. Manage Threads
    4. Manage File System
12. Input / Output
    1. Console
    2. Keyboard
    3. Mouse
    4. Video
       1. VGA
       2. Vesa
13. Files and File Systems
14. Beyond or MISC
    1. PCI
    2. Plug and Play

**Parts of OS Development (biased toward x86)**

Intro

What is an OS

Why develop an OS

Developing for x86

<http://en.wikipedia.org/wiki/X86_instruction_listings>

Develop on Windows (use PE format)

<http://ksrenevasan.blogspot.com/2005/10/writing-multiboot-pe-kernels-using.html>

Suggestions: <http://www.osdever.net/tutorials/mysuggestions.php>

Tools

Assembler, Compiler,

**Step By Step Process I went through**

1. Setup development environment.
2. Setup testing Virtual Machine.
   1. I decided to create a virtual machine for my compiling and linking environment.
   2. Write the code on Windows XP using text editor (Notepad++).
   3. FTP over to my ubuntu Server development virtual machine
   4. Copy the final compiled binary onto a floppy image file.
   5. Start Virtual Machine used to test the OS binary from the same floppy image.
3. Initial Tool Chain Test
   1. Create a NASM assembly file we’ll call ‘test.s’
4. Boot Loader
   1. I chose to use a Boot Loader called GRUB instead of “rolling” my own. However there are some great tutorials on creating a boot loader, however most people suggest that a boot loader can be just as tough as building a simple kernel or operating system.
   2. I would say that a boot loader specific to your kernel and architecture is not as difficult as developing the OS kernel, but is still quite hard to do.
5. Use a Multiboot compatible Boot Loader
   1. I used GRUB, it seems to be the most powerful and is easy to use.
6. Create GRUB based floppy image
7. Create Initial GRUB based kernel
8. Test GRUB based kernel (kernel 0.2)

**Information about the areas of OS development**

Boot Loader

Roll Your Own (RYO)

Utilize a Multi-Boot-Loader like GRUB, LILO, etc

Real Mode

Protected Mode

: <http://linuxgazette.net/issue82/raghu.html>

Crash course in pmode: <http://www.geocities.com/SiliconValley/2151/pmode.html>

Virtual Memory

Segmentation

Global Descriptor Table (GDT)

http://en.wikibooks.org/wiki/X86\_Assembly/Global\_Descriptor\_Table

Local Descriptor Table (LDT)

Interrupt Descriptor Table (IDT)

Memory Management

GRUB hands off the size of RAM in a multi-boot header

<http://www.osdev.org/osfaq2/index.php/How%20do%20I%20determine%20the%20amount%20of%20RAM%3F>

Start by writing a flat memory model using the simple GDT of one code segment (4GB) and one data segment (4GB).

<http://en.wikipedia.org/wiki/Memory_manager>

<http://www.osdever.net/tutorials/memory1.php>  
<http://www.osdever.net/tutorials/memory2.php>

Multitasking

http://www.osdever.net/tutorials/multitasking.php

Video

Standard VGA address: 0xb8000 (using standard segmenting 0xb800:0)

Real Mode Interrupt Functions

My walkthrough building Tranix!

Tips/Problems

=============

Prob:

undefined reference to '\_\_stack\_chk\_fail'

Desc:

This problem occurs when the gcc compiler thinks you have a possible buffer

overflow, or the code cannot be statically proven to be safe to such a bug.

Soln:

add -fno-stack-protector to the CFLAGS as an argument to gcc

<http://hackinglinux.blogspot.com/2006/11/resolving-stackchkfail-error.html>

-------------

Prob:

Desc:

Soln: