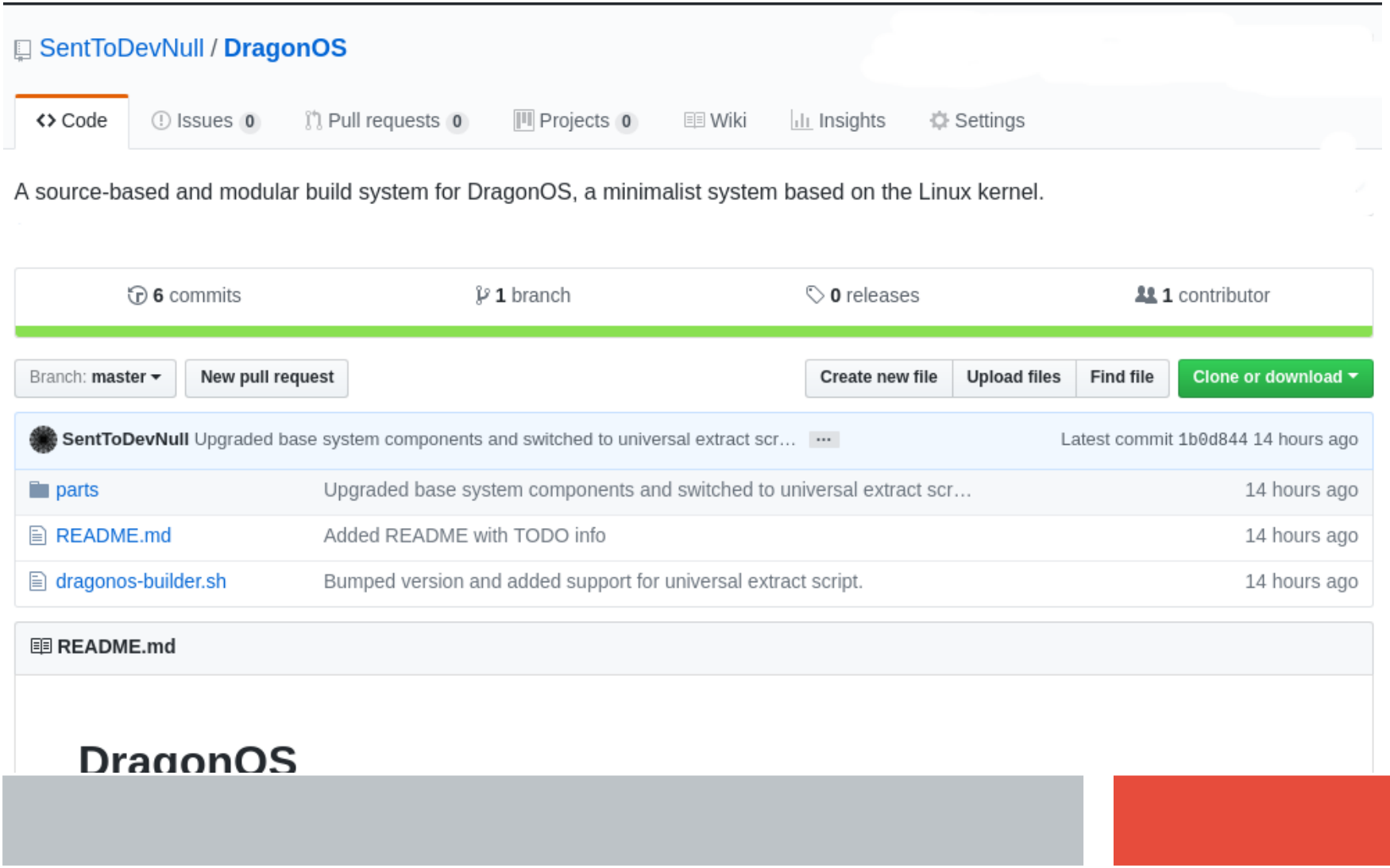
How Operating Systems Work, Why You Should Build Your Own, and How to Do It

by Lukas Yoder

My Pet Project: DragonOS

If you want, you can clone the repo and build the OS during this presentation.
Link: https://github.com/SentToDevNull/DragonOS



How I Got Started with Working on This

The Problem

DontBreakDebian

Advice For New Users On Not Breaking Their Debian System

Debian is a robust and reliable system, but it's still very easy for new users to break their systems by not doing things the Debian way. This page lists common mistakes made by new users. Some of the things listed here can be done safely, but only if you have enough experience to know how to fix your system when things go wrong.

The general theme to the advice here is that consequences are not always immediate, and can make future upgrades impossible without a complete reinstall. If upgrading without a complete reinstall is important to you, be careful not to make the mistakes outlined below.

One of the primary advantages of Debian is its central repository with thousands of software packages. If you're coming to Debian from another operating system, you might be used to installing software that you find on random websites. On Debian **installing software from random websites is a bad habit**. It's always better to use software from the official Debian repositories if at all possible. The packages in the Debian repositories are known to work well and install properly. Only using software from the Debian repositories is also much safer than installing from random websites which could bundle malware and other security risks.

Don't make a FrankenDebian

Debian Stable should not be combined with other releases. If you're trying to install software that isn't available in the current Debian Stable release, it's not a good idea to add repositories for other Debian releases. The problems might not happen right away, but the next time you install updates.

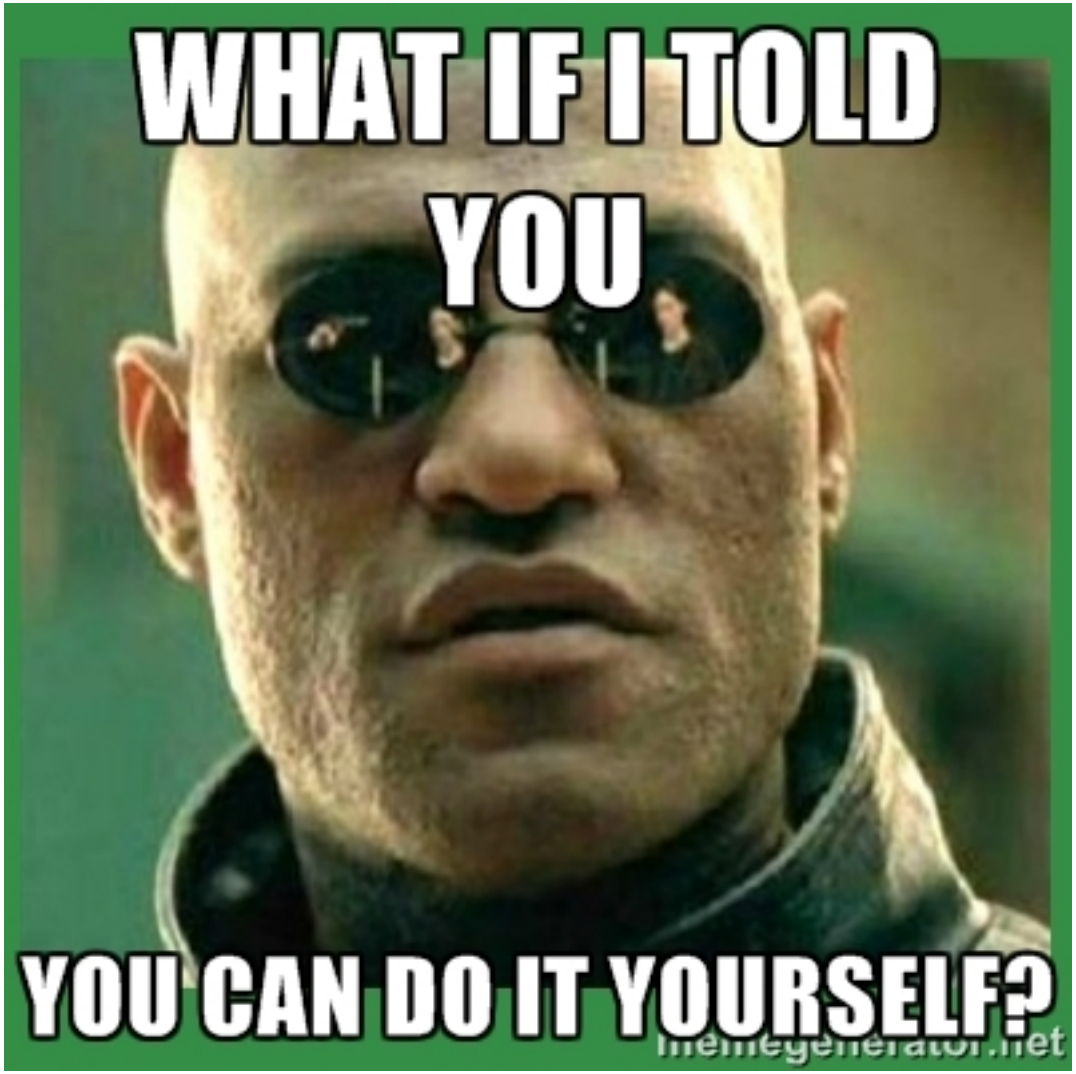
The reason things can break is because the software packaged for one Debian release is built to be compatible with the rest of the software for that release. For example, installing packages from *buster* on a *stretch* system could also install newer versions of core libraries including *libc6*. This results in a system that is not *testing* or *stable* but a broken mix of the two.

Repositories that can create a FrankenDebian if used with Debian Stable:

- Debian *testing* release (currently *buster*)
- Debian *unstable* release (also known as *sid*)
- Ubuntu, Mint or other derivative repositories are **not** compatible with Debian!
- Ubuntu PPAs

The Problem, Cont.

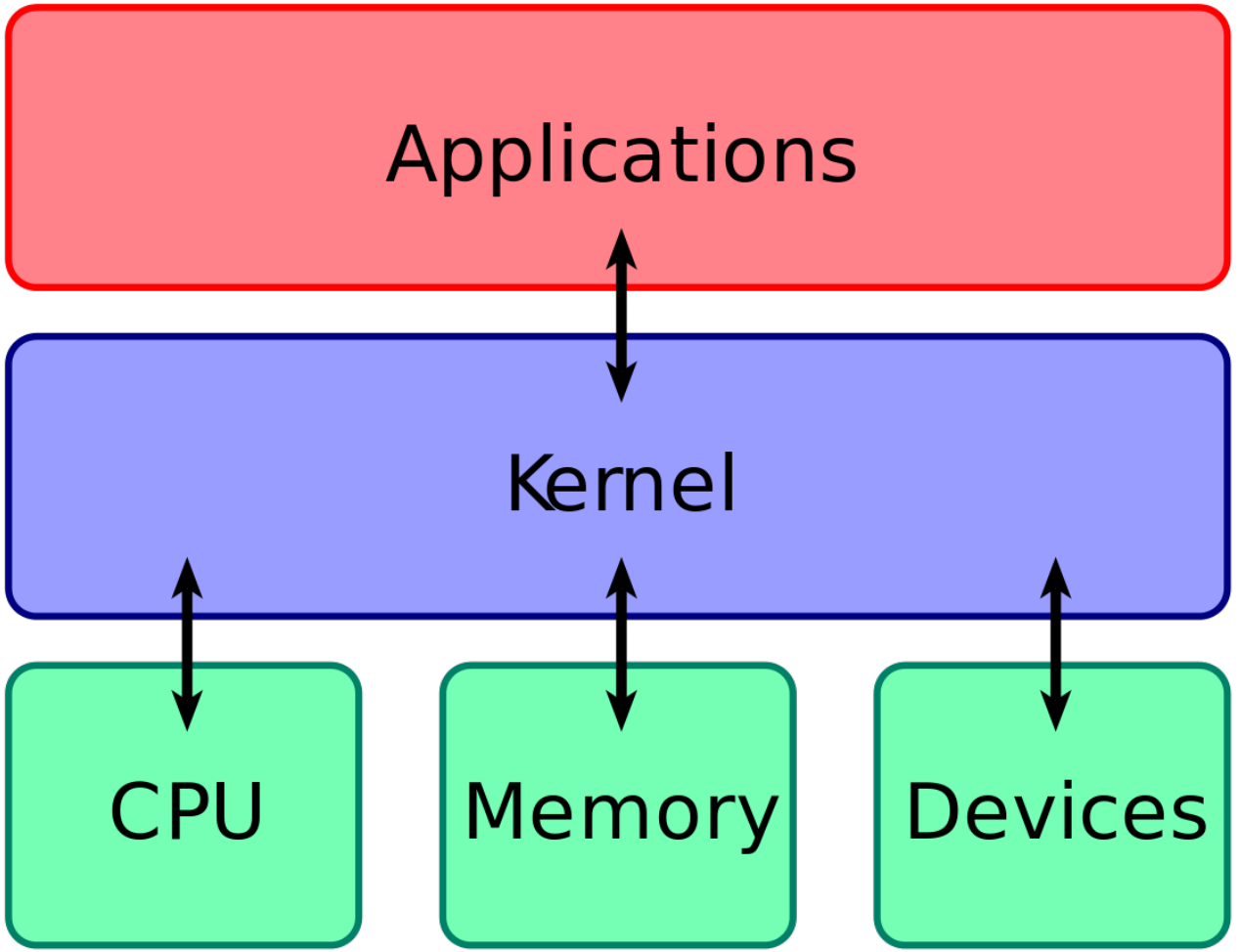
Solution: Do it Yourself



...


DO YOUR OWN PLUMBING
THEY SAID
IT'S EASY THEY SAID

How Operating Systems Work: Part 1 – The Kernel



See How Kernels Work: A Hello World Kernel

kernel.asm

```
;architecture
bits 64

;virtual address space
section .text
 ;multiboot spec mandates the following defined within the first 8kb:

 align 4 ;have a 4-byte boundary for instructions

 ;a magic number to identify the header (occupying first 4 bytes)
 ;  using "dd" for double-word (4 bytes)
 dd 0x1BADB002 ;0x1BADB002 is the magic number for a multiboot header

 ;a "flags" field occupying the next 4 bytes
 ;if bit 0 is set, all boot modules are loaded aligned on page boundaries
 ;if bit 1 is set, info on available memory must be included
 ;if bit 2 is set, info about video mode table must be included
 dd 0x00000000 ;setting bit 0

 ;a "checksum" field containing a uint32 that when added to "magic" and "flags" must equal 0
 dd - (0x1BADB002 + 0x00000000)

 ;end of multiboot section

 extern function ;the function called "function" that is defined externally...

 global start ;need to declare the start segment globally so the linker can find it
 start:
  mov esp, this_is_your_stack ;move the extended stack pointer to your stack
  call function ;self-explanatory
  hlt ;let's prevent a stack overflow; if I were decent at assembly, this wouldn't be necessary

section .bss ; your friendly neighborhood data segment for statically-allocated variables
 this_is_your_stack: resb 8192 ;reserving 8192 bytes for the stack
```

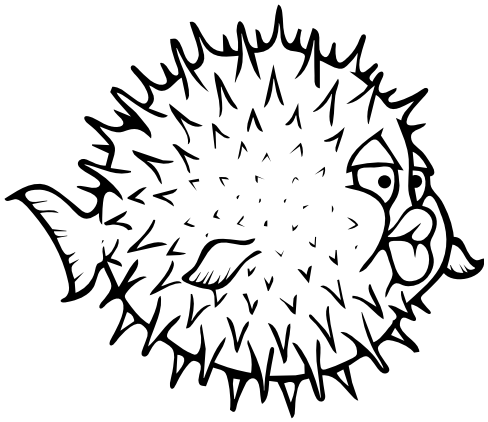
kernel.c

```
void function(void)
{
const char *str = "Heya there, world!";
//video memory begins at address 0xb8000
char *vidptr = (char*)0xb8000;
unsigned int i = 0;
unsigned int j = 0;
unsigned int screensize;

//this loops clears the screen
//there are 25 lines each of 80 columns; each element takes 2 bytes
screensize = 80 * 25 * 2;
while (j < screensize) {
//blank character
vidptr[j] = ' ';
//attribute-byte
vidptr[j+1] = 0x07;
j = j + 2;
}

j = 0;

//this loop writes the string to video memory
while (str[j] != '\0') {
//the character's ascii
vidptr[i] = str[j];
//attribute-byte: give character black bg and light grey fg
vidptr[i+1] = 0x07;
++j;
i = i + 2;
}

return;
}
```

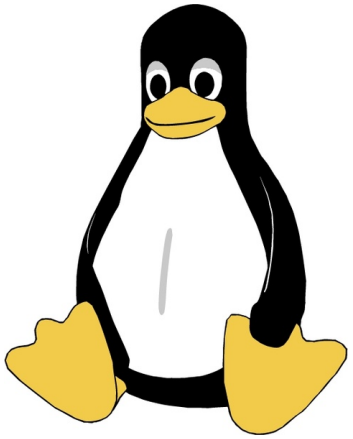

The takeaway: a lot of hard-coding values and no using C libraries like stdio.

A Buffet of Kernels

Strength: Most Widespread; on Every Motherboard with Intel Chipset; Runs the Management Engine



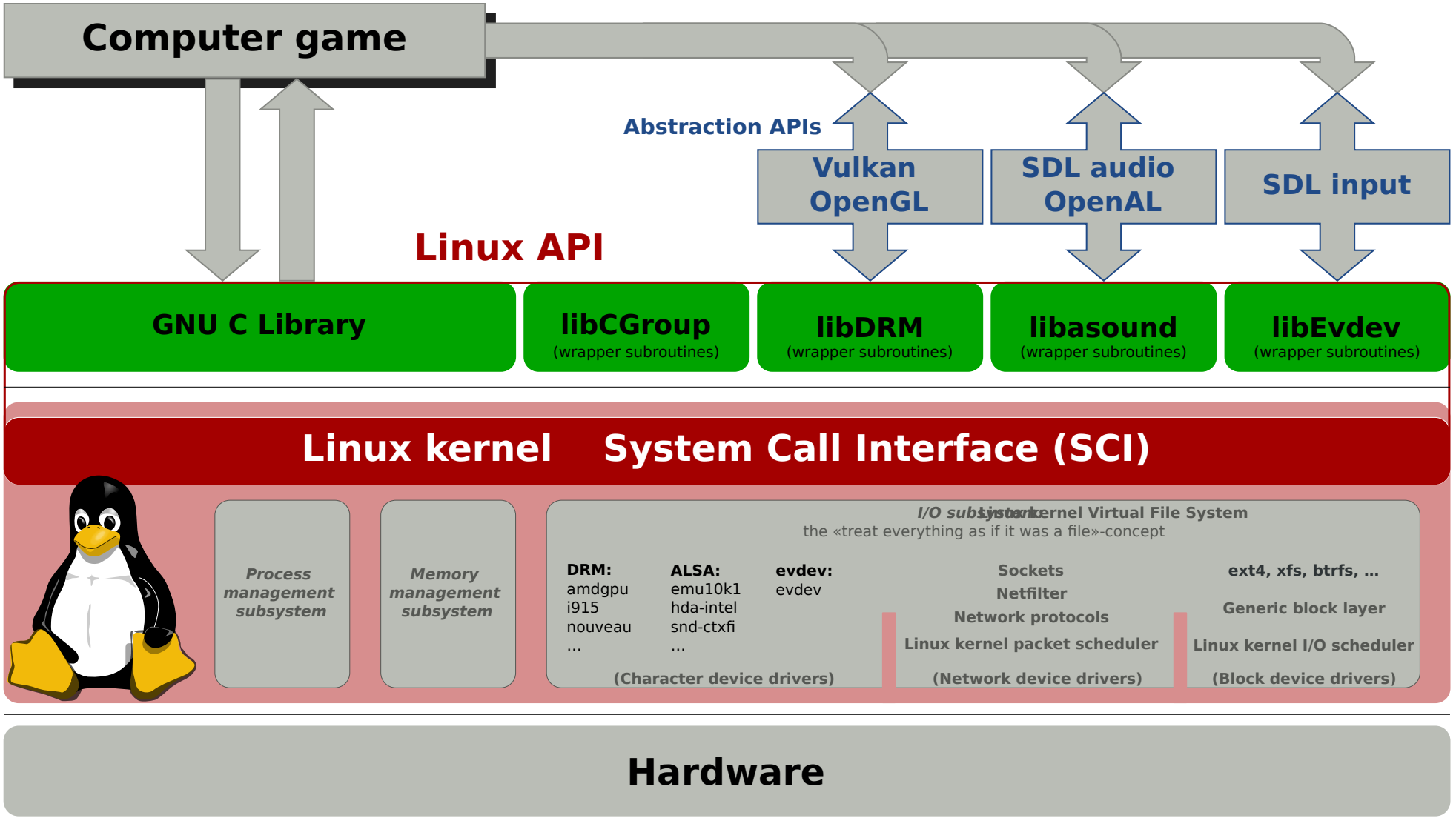
Strength: Most Paranoidly Secure; Not the Best Performance



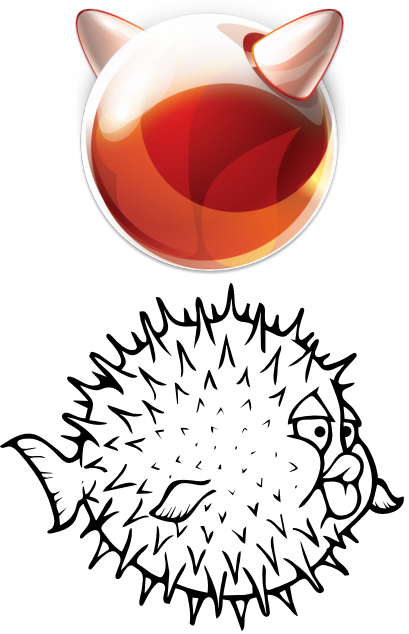
Strength: Most Hardware Support; Good Middle Ground b/w Performance & Security

Strength: Best Performance; Least Security

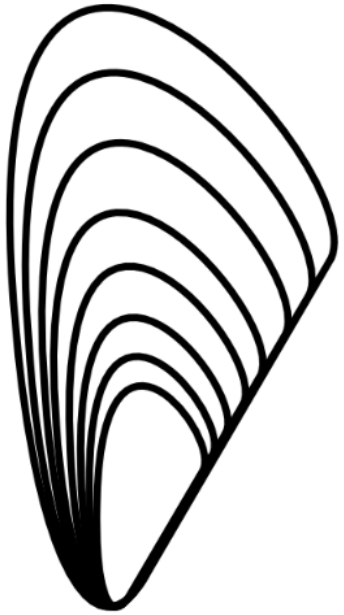
Climbing up the Ladder of Abstraction: C Libraries



Can you C all These Different Libraries?



BSD Libc: Each BSD maintains their own; Android uses a fork OpenBSD's called Bionic



musl libc: a lightweight, fast, simple, and free libc implementation; does pretty much everything glibc does with less code, less vulnerabilities, and better performance

Glibc: The GNU project's libc implementation; has been around forever; most widely used; most features; also most bloated :'(

Climbing Higher Up the Ladder of Abstraction: System Utilities

Most Important Utility: Your CLI Shell; it's a language and an interface to all programs you have installed on your system; altertatives: ZSH, KSH, CSH, DASH

```
89  #------------------------------------------------------------------------#
90  #                                                                        #
91  #        Set up a bootstrapped environment to build the system...        #
92  #                                                                        #
93  #------------------------------------------------------------------------#
94
95  function BOOTSTRAPSETUP {
96  cp parts/04-bootstrap-setup.sh $LFS
97  chown -v lfs /mnt/lfs/tools
98  chown -v lfs /mnt/lfs/sources
99  cp parts/extract.sh $LFS/sources/
100 time su lfs -c /bin/bash << "EOF"
101 exec env HOME=$HOME TERM=$TERM PS1='\u:\w\$ ' LFS=/mnt/lfs LC_ALL=POSIX  \
102   LFS_TGT=$(uname -m)-lfs-linux-gnu PATH=/tools/bin:/bin:/usr/bin bash   \
103   /mnt/lfs/04-bootstrap-setup.sh
104 EOF
105 rm -f $LFS/04-bootstrap-setup.sh
106 }
```

… and GitHub messed up the coloring in its code viewer...

Climbing Higher Up the Ladder of Abstraction: More System Utilities

```
zlib: a data compression library and utility
file: determines the filetype of a program
readline: line editing library that allows you to
          do things like move your text cursor
M4: a macro processor
bc: a “basic calculator”
binutils: programming tools for creating and managing
          binary programs, object files, libraries,
          profile data, and assembly code
GMP: a multiple precision arithmetic library
MPFR: an arbitrary-precision binary floating-point
      computation library with correct rounding
MPC: a library for the arithmetic of complex numbers
                        ...
```

```
ncurses: a text-based user interface library
bzip2: another compression algorithm
GCC: the GNU compiler collection, which contains compilers
     for C, C++, Objective-C, Objective-C++, Fortran, Java,
     Ada, Go, etc.
sed: a stream editor; used a lot for text manipulation
perl: a high-level interpreted language
coreutils: software that implements many basic unix tools
util-linux: software implementing over 100 basic linux
            system utilities not in coreutils
GRUB: GNU’s Grand Unified Bootloader
man: a utility that helps you browse manual pages
vim: the one true text editor
                        ... and many more...
```

Note 1: A lot of these tools don’t just have 1 implementation, but rather many different ones. For example, GNU Binutils is not the only binutils implementation. Some of these utilities can be substituted for others that perform similarly, but a functional system requires all of these.

Note 2: Some of the utilities here are deemed necessary by the POSIX (Portable Operating System Interface) standard.

Note 3: These aren’t necessarily the names of command-line utilities, but rather pieces of software that may consist of more than 1 program.

Why is all this necessary?

Abstraction like this is necessary so that you don't have to worry about buffers, registers, and CPU microarchitectures to do basic tasks.

```
[user@system ~]$ echo 1+2 | bc
3
```

Climbing Higher Up the Ladder of Abstraction: Init Systems

Init systems are the first processes started when operating systems boot. They manage other system processes. (For instance, your network service that connects you to WiFi after you turn your computer on.)

There are many different init systems to choose from, including:

- Systemd
- Launchd
- System V
- Upstart

Different init systems have different strengths and weaknesses. For example, systemd is very fast compared to System V, which used to have more widespread use, but System V is preferred by some because it follows the Unix philosophy by not being monolithic in design.

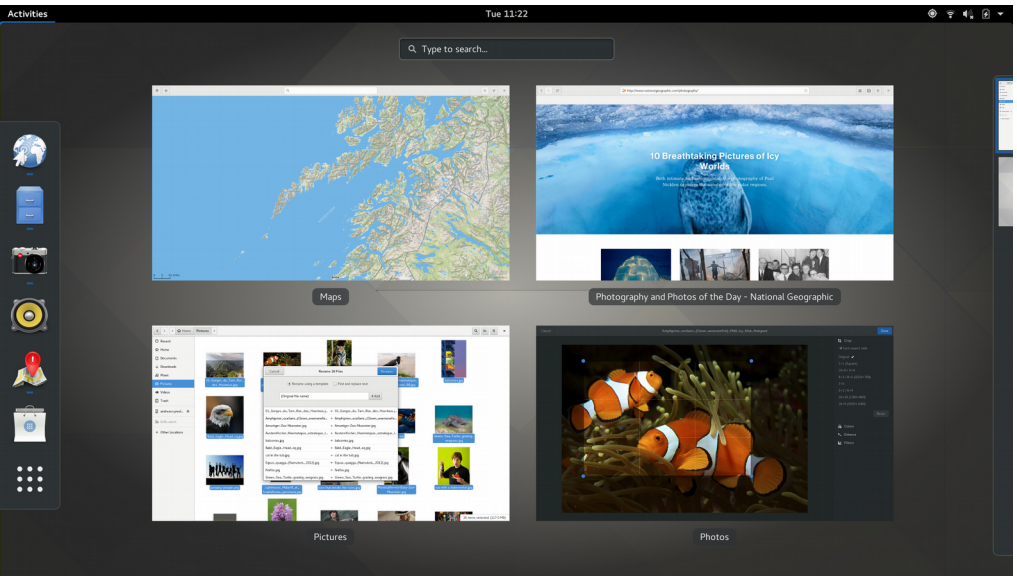
Climbing Higher Up the Ladder of Abstraction: Display Servers

X.Org

Wayland; the successor of the X.Org project, intended to be more secure, among other things

Climbing Higher Up the Ladder of Abstraction: Graphical Shells

Sometimes you don’t need the power of a CLI shell and it’s just easier to point and click.



GNOME 3

KDE Plasma 5

Building Your Own System

Not easy… Occasionally not very fun…

Some Challenges:

- Software components don't always play nice with each other.
  - ex: the more recent version of util-linux doesn't seem to compile with ncurses, an optional dependency
- Sometimes base system software requires patches to run on your platform. (Sometimes it needs patches to function at all.)

Something like the Linux From Scratch Project can provide a nice base for your operating system, since it records what software is compatible with other software and what needs to be done to make certain software compile and work with your system.

Linux From Scratch Project

If you're interested in operating system development, the LFS manual is a great way to learn about ONE way to do it.

If you want to make a system that you can actually maintain and add to, the LFS project's instructions aren't the best way to go.

- Take the basic concepts and implement them the way you feel is best.
- The way LFS structures the steps in their manual is to maximize educational value, not to be the most efficient. (That's even stated somewhere in the documentation, if you can manage to find it.)
  - For example, the way that the LFS project is structured prevents automation through the shell. (Indeed, their automated LFS project has to use a Java program to parse through the PDF textbook and use system libraries to build their OS.)

Things That Are Probably Useful to Implement

- A package manager to keep track of the software you build.
- Automated logging to enable you to resume builds if they error out at any point (or if the power to your PC shuts off).
- Automation of everything that the LFS manual wants you to do manually, such as checking software on the host system.
- Writing to an ISO file instead of an actual partition.
- A build system that enables you to easily debug problems with your build and restart or continue it if necessary.
- A package cache so that you don't have to keep re-downloading files.
- Any changes you feel make the system more awesome.

… and that's where my pet project comes into play

Among other things, it adds the stuff discussed in the last slide to make development easier.

If you're interested in building your own unique system, you can read the LFS manual, read these slides, and get started hacking.

If you just want to test a system like this out, you can clone my repo (https://github.com/SentToDevNull/DragonOS), run the build script, install (development versions of) dependencies, and have fun.

Now, A Demo of DragonOS