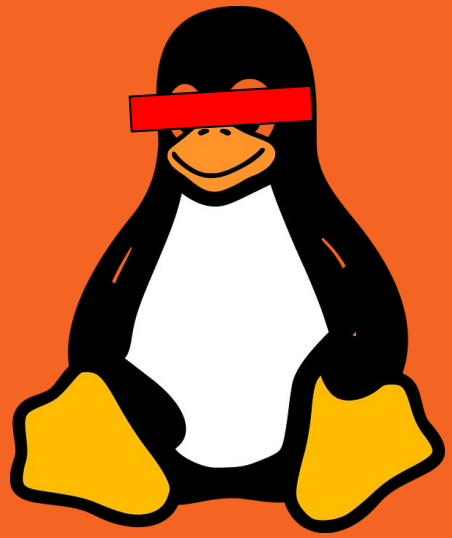
The Intricacies of Sizecoding on Linux



Pushing the interest and boundaries of a very interesting platform

Shiz/WRD PoroCYon/K2

Ew, Linux?!


Pick one or more
• Windows
• Linux
• MacOS
• Amigaaaaaaa
• C64
• A CPU and some RAM
Pick one or more
• Windows ← You are here
• Linux
• MacOS
• Amigaaaaaaa
• C64
• A CPU and some RAM
• Something else!


Pick one or more
• Windows
• Linux
• MacOS
• Amigaaaaaaa
• C64
• A CPU and some RAM
• Something else!
PROJECTION SCREEN

Ew, Linux?!

People are interested!

What makes Linux interesting?

➔ **Libraries!**

FFT? Sure!

GLU, Opus, all there!

Rendering? ;-)

http://releases.ubuntu.com/18.10/ubuntu-18.10-desktop-amd64.manifest

```
mark@nene: ~
% curl -s http://releases.ubuntu.com/18.10/ubuntu-18.10-desktop-amd64.manifest
 | wc -l
1785
% curl -s http://releases.ubuntu.com/18.10/ubuntu-18.10-desktop-amd64.manifest
 | grep ^lib | wc -l
826
%
```

Ew, Linux?!

People are interested!

What makes Linux interesting?

➔ **Libraries!**

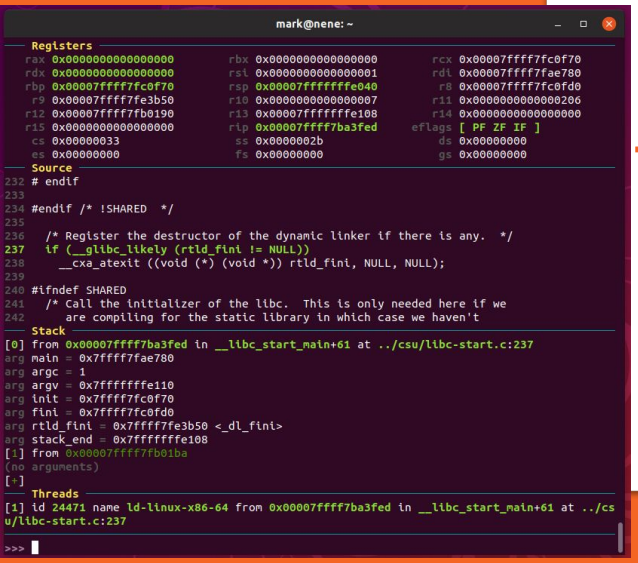
FFT? Sure!

GLU, Opus, all there!

Rendering? ;-)

➔ **Open!**

See what your dependencies *really* are doing!



Ew, Linux?!

People are interested!
What makes Linux interesting?

- ➔ **Libraries!**
  FFT? Sure!
  GLU, Opus, all there!
  Rendering? ;-)
- ➔ **Open!**
  See what your dependencies *really* are doing!
- ➔ **Universal!**
  Secretly powers a lot of devices, lots of opportunities!

Linux prod count:

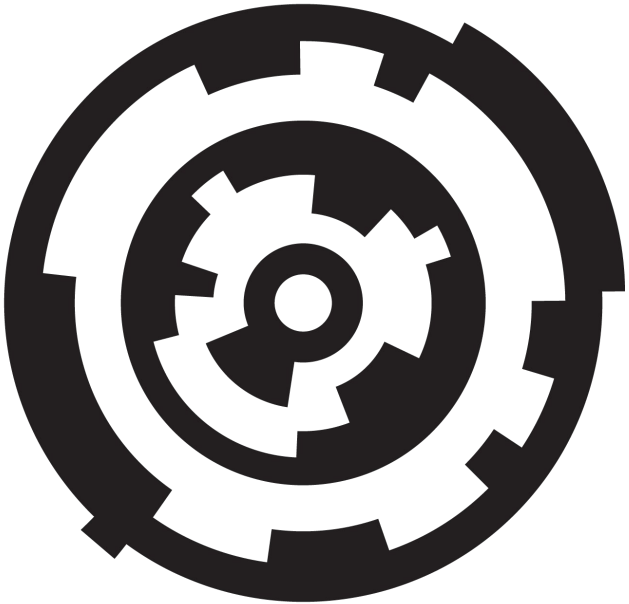
?

Linux prod count:

WHAT'S WRONG?

(as per Pouet)

OFFICIAL™ DEMOPARTY SURVEY



1

LACK OF TOOLING

2

LACK OF INFO

3

(PRECONCEPTIONS?)

Revision party quotes

"Linux sucks! But every OS sucks."

"I wouldn't even know where to start..."

"I'm giving up on this piece of shit VST, I'll just do Famitracker instead"

Quotes for illustrative purposes only, possibly slightly altered to protect the guilty

```
# systemctl enable linux-prods

# rc-update add linux-prods

# update-rc.d add linux-prods

# chkconfig --add linux-prods
```

. TOC .

1. Target platform

2. Executable basics
3. Dynamic linking
4. Console to GL
5. Crunching it all down

6. The ***FUTURE*** »

$ stat /target

- ➔ Latest Ubuntu 64-bit
  currently: **18.10**
- ➔ Proprietary NVIDIA driver
- ➔ Default packages **only!**

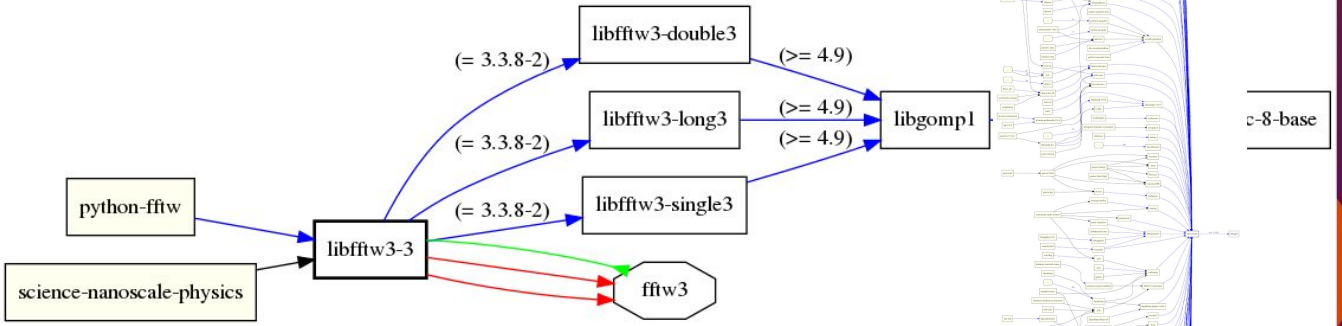
(still a lot)
libopus, imagemagick, fftw, espeak...

Consequences

- ➔ glibc `ld.so` (dynamic linker)
- ➔ no SDL, GLFW, GLUT
- ➔ 64-bit libraries **only**
  … no x32 either … :-(
- ➔ Static linking: forget about it

Consequences

- **But!** Lots of inte
  already there
- Fourier, text-to-
- Your **own** discretion what to use and what not!
- Things may ***disappear***
- **`apt-rdepends -d --state-follow=Installed <pkg>`** can be used to figure out dependencies**!**



Loading ELF

- ELF headers can be **big!**
- Who checks these?

Static: **kernel**
Dynamic: **ld.so**

```
typedef struct {
    unsigned char e_ident[EI_NIDENT];
    Elf32_Half    e_type, e_machine;
    Elf32_Word    e_version;
    Elf32_Addr    e_entry;
                  off, e_shoff;
                  ags;
                  size;
                  entsize, e_phnum;
                  entsize, e_shnum;
                  strndx;
```

```
typedef struct {
    Elf32_Word    p_type;
    Elf32_Off     p_offset;
    Elf32_Addr    p_vaddr;
                  ddr;
                  lesz;
                  msz;
                  ags;
                  ign;
```

```
typedef struct {
    Elf32_Word    st_name;
    Elf32_Addr    st_value;
    Elf32_Word    st_size;
    unsigned char st_info;
    unsigned char st_other;
    Elf32_Half    st_shndx;
} Elf32_Sym;
```

Does it need all of this?

```
$ readelf -a ./test
  Class:                             unknown: 31
  Data:                              unknown: db
  OS/ABI:                            unknown: b0
...
  Start of section headers:          23378022 (bytes into file)
  Flags:                             0x6ebe389
...
  Size of section headers:           32973 (bytes)
  Number of section headers:         60248
```

```
$ ./test
hello world
```

Process loading examined

(how do I shot ELF?)

System calls!

Kernel decides what happens:

➔ **Static binary:** kernel parses directly

➔ **Dynamic binary:** dynamic loader loads binary instead

Check parsing code of **either**!

➔ Unparsed fields → code (or zeroes)

➔ Parsed but controllable fields (`e_entry`)

Stack layout:

```
          argv[0] = "./test"      <----.
rsp-n-m:  AT_NULL, 0                   |
rsp-n   : AT_FOO, bar                  |
          NULL                         |
          ...                          |
          environ[1]                   |
rsp-48:   environ[0]                   |
          &argv[3] = NULL              |
          &argv[2]                     |
          &argv[1]                     |
          &argv[0]     ----------------'
rsp:      argc=3
```

Entry to sizecoding

- ➔ Default entrypoint:
  C runtime library (`crt0/crt1.o`)
- ➔ Roll our own! `-nostartfiles`
- ➔ Info passed by kernel on the stack

Stack not aligned by default!

Required by SSE or segfault
1-byte trick: **`push rax`**

- ➔ Manual system calls (`int 0x80/syscall/swi #0`)
- ➔ Graphics: `/dev/console`, `/dev/vcsa`, `fbdev`
  https://www.pouet.net/prod.php?which=3696
- ➔ Sound: device files, `ioctl()`, `write()`
  nah actually, just pipe to `/usr/bin/aplay`

Meet Marcos.

- ➔ Serious programmer
- ➔ Runs a Windows shop
- ➔ Doesn't know dynamic linking
- ➔ Wants to learn Linux sizecoding

Story for illustration purposes only

Marcos just wants to use GL.

What he needs is...

Dynamic Linking

...but he's heard that's hard.

What makes dynamic linking so heavy?

```
typedef struct {
    Elf64_Word    st_name;
    unsigned char st_info;
    unsigned char st_other;
    Elf64_Section st_shndx;
    Elf64_Addr    st_value;
    Elf64_Xword   st_size;
} Elf64_Sym;
```

- ➔ Dynamic linker (`PT_INTERP`)
- ➔ Needed libraries (`DT_NEEDED`)
- ➔ All symbols, verbatim
- ➔ PIC and relocation headers
- ➔ Support headers

```
$ cc -Os -o hello hello.c && strip hello
$ ls -l hello
-rwxr-xr-x  1 mark mark    14312 Apr 20 09:58 hello
```

What makes dynamic linking so heavy?

Relocation: dynamic linker instruction to patch an executable in-memory

Everytime an instruction accesses a symbol.
This scales absolutely terribly.

GOT: *global offset table*, structure at the start where all resolved symbols are put *once*
PLT: *procedure linkage table*, small code stubs that jump to addresses in the GOT

```
typedef struct {
    Elf32_Addr r_offset;
    Elf32_Word r_info;
} Elf32_Rel;

typedef struct {
    Elf32_Addr  r_offset;
    Elf32_Word  r_info;
    Elf32_Sword r_addend;
} Elf32_Rela;
```

```
61  if (elf_header->e_ident [EI_CLASS] != ElfW (CLASS))
62    {
63      if (opt_verbose)
64        {
65          if (elf_header
66            error (0, 0,
67          else if (elf_h
68            error (0, 0,
69          else
70            error (0, 0,
71        }
72      return 1;
73    }
```

Import-by-hash!

ld.so cares more :(

- ➔ Need dynamic linker to do anything at all
- ➔ Extremely minimal headers become hard
  l non-standard

- ➔ Nobody said we had to use `ld.so` *all the time*!
- ➔ Minimal but *barely-satisfying* headers to just have `ld.so` load our binary and dependencies
- ➔ Do everything else ourselves, **no more Elf_Sym/Elf_Rel headers and extra sections**!

```
$ ld ./test
Dynamic section at offset 0xe0 contains 6 entries:
  Tag        Type
Name/Value
 0x0000000000000005 (STRTAB)             0x10164
 0x0000000000000001 (NEEDED)             Shared
library: [libGL.so.1]
 0x0000000000000001 (NEEDED)             Shared
library: [libgtk-3.so.0]
 0x0000000000000001 (NEEDED)             Shared
library: [libgobject-2.0.so.0]
 0x0000000000000001 (NEEDED)             Shared
library: [libc.so.6]
 0x0000000000000006 (SYMTAB)             0x0
```

Prior art

➔ **bold**
http://www.alrj.org/pages/bold.html

Paved the way for this approach!
Processes relocations manually
Not portable
Broken due compiler change :-(

➔ **dnload**
https://github.com/faemiyah/dnload

Portable!
Includes GLSL minifier
Own full linking step
Very intrusive
Hardcoded list of known symbols
Input is (C/C++) source, not object files
Was until recently broken on Linux

introducing smol

simple/shoddy/smart minsize-oriented linker

simple	simply replace your `ld` step with `smold`
portable	wraps GNU ld, loader code portable to any ELF platform
generic	feed it `.o` files from any language, integrate it how you want
smaller	uses custom hash-based import system, no `STRTAB/SYMTAB`

https://github.com/Shizmob/smol

[light side]

1. scan input `*.o` files for unresolved symbols

   `readelf` & `scanelf` to the rescue!

2. resolve symbols
   create hash symbol table

   we have all the required flags!

3. create minimal new header
   attach loader, do final link

   custom linker script for efficiency

```
smold$ readelf -sW obj/hello.start.o | grep UND
NOTYPE  LOCAL  DEFAULT  UND
NOTYPE  GLOBAL DEFAULT  UND __printf_chk
NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE

NOTYPE  GLOBAL DEFAULT  UND exit
NOTYPE  GLOBAL DEFAULT  UND __libc_start_main
smold$
```

```
[section .rodata.neededlibs]
_strtab:
        _symbols.libc: db "libc.so.6",0
[section .data.smolgot]
_symbols:
global __printf_chk
__printf_chk:
                _symbols.libc.__printf_chk: dq 0xb4ebe20b
global __libc_start_main
__libc_start_main:
                _symbols.libc.__libc_start_main: dq 0xf63d4e2e
global exit
exit:
                _symbols.libc.exit: dq 0x7c967e3f
db 0
_symbols.end:
```

[dark side]

idea: obtain dynamic linker's own state and abuse it for symbol resolution

➔ `DT_DEBUG` section, offset `0x4`
meh, extra sections
➔ leak from `_dl_start_user()`

poke **`ld.so`** state, obtain library symbol hash arrays **for free**!

load

```
/* Symbol hash table.  */
Elf_Symndx l_nbuckets;
Elf32_Word l_gnu_bitmask_idxbits;
Elf32_Word l_gnu_shift;
const ElfW(Addr) *l_gnu_bitmask;
union
{
  const Elf32_Word *l_gnu_buckets;
  const Elf_Symndx *l_chain;
};
union
{
  const Elf32_Word *l_gnu_chain_zero;
  const Elf_Symndx *l_buckets;
};
```

x86_64: own mini-PLT+GOT

x86: `E9 D00DBEEF`

jump! hash replaced by relative address
PLT *is* GOT

Other small tricks

- ➔ `PT_INTERP`:
  do we *actually* need it?
  `$ /lib64/ld-*.so ./x`

- ➔ **Sections?** Never heard of ‘em!
  one giant `RWX` übersection

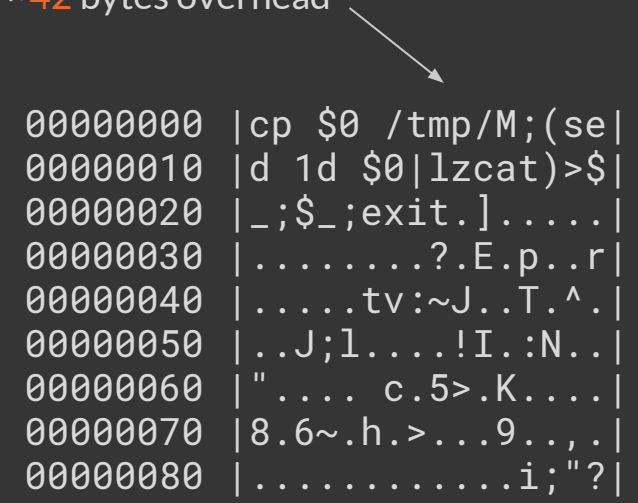
- ➔ `argc/argv/envp`
  not passing them saves bytes!
  bring your **own** environment

Speaking of *small* tricks...

COMPRESSION.xz

Everyone uses shell droppers

~42 bytes overhead



```
00000000 |cp $0 /tmp/M;(se|
00000010 |d 1d $0|lzcat)>$|
00000020 |_;$_;exit.]....._|
00000030 |........?.E.p..r|
00000040 |.....tv:~J..T.^.|
00000050 |..J;l....!I.:N..|
00000060 |".... c.5>.K....|
00000070 |8.6~.h.>...9..,.|
00000080 |............i;"?|
```

—

- Touch the file system
- *Ewww*, sh?!
- LZ compression sucks (for x86):

But they *suck*.

```
8d3d1c060001    lea edi, [0x100061c]
8b2db8000001    mov ebp, [0x10000b8]
8b6d04          mov ebp, [ebp + 4]
8b6d0c          mov ebp, [ebp + 0xc]
8b7508          mov esi, [ebp + 8]
ad              lodsd eax, [esi]
83f805          cmp eax, 5
75fa            jne 0x1000102
```

Attempt to fix #1:

Fishypack, vondehi

- ➔ Really small decompression stub (asm)
- ➔ Does everything in-memory (`memfd_create`)
- ➔ A bit larger: ~160 bytes
- ➔ Still LZ (uses `gzip/xz`)

https://gitlab.com/PoroCYon/vondehi

Attempt to fix #2:
XLINK

See also: LCA2019, “Executable Code Golf”

- ➔ PAQ1-based Arithmetic Range Coder
- ➔ Sparse *n*-gram model → good for x86!
- ➔ No kkrunchy “instruction stream splitting”; use general approach

Currently DOS-only…
Expect more soon!

https://github.com/negge/xlink

Compiler voodoo

Telling gcc/clang to behave when you don't want to go 100% ASM

Compile

```
-O2/-Os -march=haswell -ffast-math

-flto -fuse-linker-plugin
-fno-unwind-tables
-fno-asynchronous-unwind-tables
-ffunction-sections -fdata-sections

-no-pie -fno-pic -fno-pie -fno-plt
-fno-stack-check -fno-stack-protector
-fomit-frame-pointer
```

Link

```
-march=haswell -Wl,-i

-flto -fuse-linker-plugin
-ffunction-sections -fdata-sections
-Wl,--gc-sections

-no-pie -fno-pic -fno-pie -fno-plt
-nostdlib -nostartfiles
```

Output: **another object file**

Now we have all the ingredients

Road to GL

Linking to GL: `GetProcAddr`

… or use `libglvnd`, link as usual

Attempt #1

X11+GLX

https://github.com/blackle/Linux-OpenGL-Examples: **xlib-opengl.c**

xlib sucks

Verbose API, brittle, GLX stuff

Attempt #1.1

X11+EGL

More compatible than GLX!
Less strict about the window state!

This still sucks.

EGL can sometimes be worse!

Attempt #2

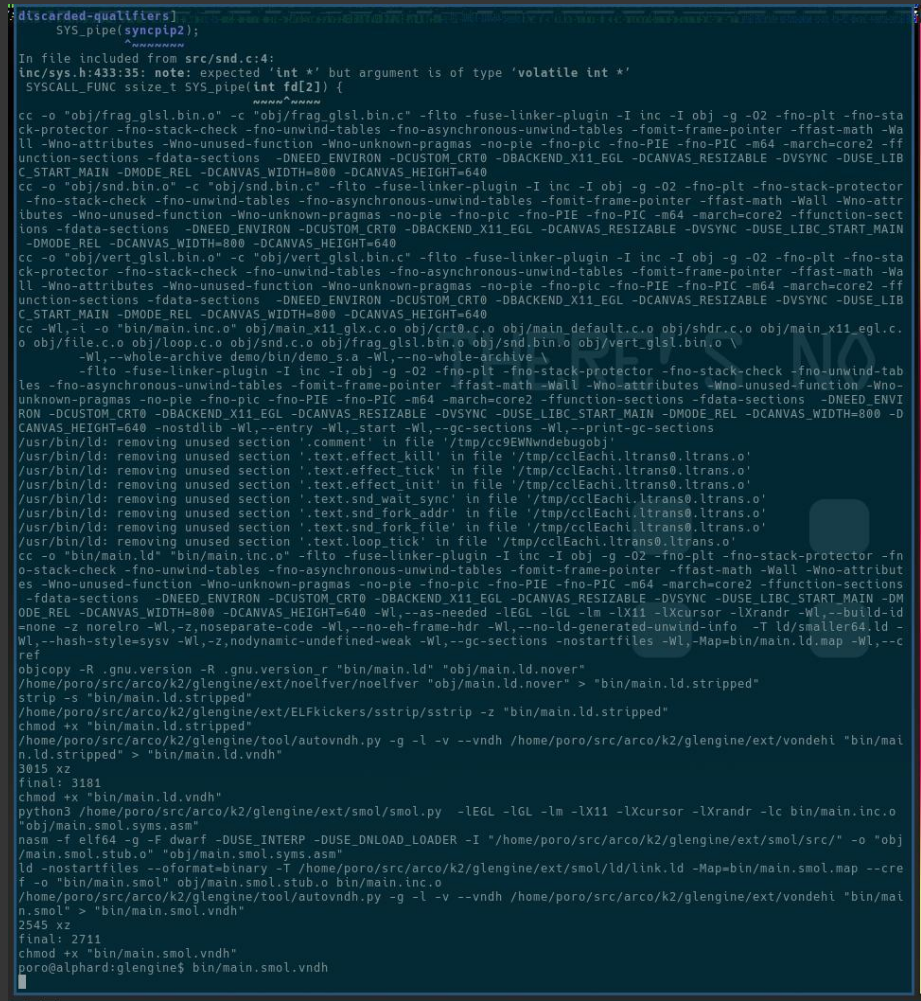
```
...
libgstreamer1.0-0:amd64 1.14.4-1
libgtk-3-0:amd64        3.24.1-1ubuntu2
libgtk-3-bin            3.24.1-1ubuntu2
...
```

ubuntu-18.10-desktop-amd64.manifest

```
gtk_gl_area_make_current(gtk_gl_area_new());
```

We have a GL context!

https://github.com/blackle/Linux-OpenGL-Examples: **gtk-opengl.c**



sound, (unoptimized) shader, no cursor, 64-bit

~2kb

smol + vondehi + lzma

Go make an intro!

Lots of optimizations are still possible, this is only the beginning

Shiz/WRD
/dev/by-row/**15**
hi@shiz.me
twitter.com/dev_console

PoroCYon/K2
/dev/by-row/**20**
no social media
just IRC

We greet

auld / alrj / blackle / breadbox / Calodox
faemiyah / gib3 & tix0 / las / leblane
parcelshit / PWP / Team210 / unlord / yx

Questions?

github.com/**Shizmob/liner**

#lsc on IRCNet