Linux 4k Intro Coding

Aug 3rd 2006

Markku “Marq” Reunanen

Contents

- Overview
- Choosing the language
- Dealing with GCC
- Dynamic library loading
- Music generation
- Compression
- Code level tricks
- Useful tools and some pointers

Overview

- Linux 4k intros have been around a few years
  - Suitable platform because of good tools, useful libraries and minimal executable overhead
- By knowing the tricks of the trade you can spend the precious 4096 for the actual content
  - You can save hundreds of bytes with a little extra work
- The methods presented here are based on the three 4k intros by me and Antti “NF” Silvast
  - Yellow Rose of Texas (Asm'03)
  - Je Regrette (Asm'04)
  - Make It 4k (Asm'05)

Choosing the language (1)

- Asm
  - You know what you're doing. No overhead but error prone and not easy to try out or tweak stuff.
- C
  - More overhead but still suitably small. Less painful than pure asm :)
- C++
  - Too much overhead for a 4k. Painful name mangling.
- Others
  - For example Perl is pretty much available everywhere. No interface to gfx/music but potential for scripting.
  - Shader asm & GLSL

Choosing the language (2)

- Our approach: combined C and asm
- System level code, soft synth and startup in asm
  - These need to be written only once
  - As small as it gets
- Effect code in C
  - Easy to code and portable
  - We were able to release Linux/Win/OSX/SGI versions in less than a week
- There's still some overhead in using C but that's the penalty of being lazy

Dealing with GCC

- GCC was the natural choice for a C compiler
  - Free, effective, available
- You can do a lot by just command line switches:
  - -Os tends to suck, -O1 usually better
  - -ffast-math of course
  - -fshort-double (dangerous!)
  - -nostdlib
  - -fno-inline, -fmove-all-movables, -fpeephole2, -fforce-mem, -fexpensive-optimizations, etc.
- There's no such thing as a perfect parameter set
- GCC version does matter!
  - By my experience 3.2 creates the smallest binaries

Dynamic library loading (1)

- Important external libraries: SDL & OpenGL
  - Some consider SDL use lame -- matter of opinion. This method is equally valid for GLX, GLUT and others.
- Using an external library function generates about 70 byte (compressed) overhead if done via standard dynamic linkage
- 1st solution: try to minimize the number of external function calls
  - For example do not use both glVertex2f and glVertex3f
- For any GL effects we need at least 10-20 functions. More tweaking required.

Dynamic library loading (2)

- Solution: open the libraries ourself and call them through function pointers
- Easy to do by using dlopen and dlsym functions
  - Open library with dlopen
  - Get pointers to functions with dlsym
  - After this they can be used from C or asm as usual
- Can be done in C but better to use asm for loading
- Overhead reduced to approximately 20 bytes (compressed) per function
- Remember to put -ldl on linker command line!

Music generation (1)

- Unfortunately, these days we need music for 4k intros too
- Under Linux no common high level sound API
  - OSS/ALSA not high-level, MIDI not common and has poor quality anyway
- Need for a soft synth
- Our solution: pure asm synth with four wave-forms, large number of channels and some effects
  - Typically takes around 1.5k (compressed) with the tune
  - Basic waveform generation and mixing easy
  - ADSR a necessity in practice

Music generation (2)

- Finally, effects make the beeps sound fat:
  - Frequency sweep, especially for bass drums
  - Amplitude modulation
  - Delay loop echo
- Not overly hard to code but does involve some effort
- Our synth is freely available -- but probably not easy to understand
- Composing for such a synth is not for the weak of heart: plain text or even asm file
  - Get a tech savvy musician or write a front-end or a converter

Compression (1)

- Gzip is available on every single Linux box, thus the well-known gzip stub compression trick:
  - The intro starts with a shell script that uncompresses and executes the following compressed binary data
  - Use gzip -n and –best for the smallest result
- Here's our attempt at a stub (56 bytes):

```
a=/tmp/I;tail -n+2 $0|zcat>$a;chmod +x $a;$a;rm $a;exit
```

- Is it really optimal?
  - Must use /tmp according to the rules
  - Executable flags must be set
  - Binary must be removed from /tmp!
  - Feel free to improve ;)

Compression (2)

- Dealing with compressed code is not always straightforward
  - Hand-tuning may actually increase the code size if it compresses less
  - The effect of locally removing or adding instructions or function calls appears pretty random
  - The same is true for compiler flags but can be helped easier. More about that later.

Code level tricks (1)

- Remove subroutines
  - Makes the code a little messier but you get rid of the entry/return instructions
- Use floats instead of doubles
  - Standard math routines use doubles and take unnecessary space unless you apply -fshort-double. Note that you can't call external functions with double parameters after this.
- Static tables
  - Declaring local arrays as static removes their init code yielding some bytes

Code level tricks (2)

- A tiny pseudorandom generator can be built with a simple rotation, xor and addition:

```
%define RANDOM_SEED 0f31782ceh

rnd:    mov     eax,[rndi]
        add     eax,RANDOM_SEED
        xor     eax,RANDOM_SEED
        ror     eax,1
        mov     [rndi],eax
        ret

rndi:   dd      RANDOM_SEED
```

Useful tools

- NASM, the Netwide Assember
  - Proper syntax, incbin, macros, free and all
- ELF kickers package and especially sstrip
  - Strips all unnecessary segments and some more out
- GC Masher
  - Helps you select an optimal set of command line parameters for GCC
  - Takes some time to brute force test a set of parameters but it's all free bytes to you
  - For example “Je Regrette” lost 74 bytes

Some further pointers

- Brian Raiter's “A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux”
  - Serious ELF header hacking for a minimal startup
- Timo Wigren's “HOWTO: 4k intros in GNU/Linux”
  - Some basic tricks for size tweaking
- Full source and Makefiles of our prods are available on the Fit homepage (http://www.kameli.net/fit/)
  - “Make it 4k” has the most recent tricks except GC Masher in the archive

The End

Thanks for your attention!
Questions? Comments?