Working on LLVM/Clang for Chrome

EDAN75
8 October 2018
hwennborg (at) google.com
Short bio

I was d04hw@efd.lth.se

Took this course in 2008

Graduated 2010, joined Google

London 2010-2013

Mountain View 2013-2017

Munich 2018-

Working on Clang/LLVM for Chrome

At the programming competition in Lund 2009 ->
Plan for this morning

1. How I ended up at Google and what I work on

break

2. How LLVM generates code for switch statements
2008: Optimizing Compilers

“Every academically educated computer scientist must know how a computer functions, and must understand the ways and methods in which programs are represented and interpreted. Compilers convert program texts into internal code. Hence they constitute the bridge between software and hardware. “

Niklaus Wirth

Compiler Construction
### 2008: Optimizing Compilers

#### Optimizing Compilers Hall of Fame at LTH

<table>
<thead>
<tr>
<th>Year</th>
<th>Group</th>
<th>Programme</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016</td>
<td>Johan Ju</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td>Karl Hylén</td>
<td>F</td>
<td>40292</td>
</tr>
<tr>
<td>2013</td>
<td>Erik Hogeman/Mads Nielsen</td>
<td>D</td>
<td>49155</td>
</tr>
<tr>
<td>2012</td>
<td>Martin Nitsche</td>
<td>Math. Göttingen</td>
<td>33526</td>
</tr>
<tr>
<td>2011</td>
<td>Linus Åkesson</td>
<td>PhD/CS</td>
<td>112805</td>
</tr>
<tr>
<td>2010</td>
<td>Joakim Andersson/Jon Steen</td>
<td>D</td>
<td>126616</td>
</tr>
<tr>
<td>2009</td>
<td>Manfred DelIkrantz/Jesper Öqvist</td>
<td>D</td>
<td>950</td>
</tr>
<tr>
<td>2008</td>
<td>Jonas Paulsson</td>
<td>D</td>
<td>18977</td>
</tr>
<tr>
<td>2007</td>
<td>Björn Carlin/Hans Gylling</td>
<td>$\pi$/D</td>
<td>1047</td>
</tr>
</tbody>
</table>
2009: Master’s Thesis at ARM in Lund

- They do compiler stuff and use LLVM
- Probably even more these days
- Graduation was getting closer
- Considered pursuing PhD but didn’t really have any ideas

“You should try interviewing at Google! The interviews are fun, and you get a free lunch if they bring you on-site.”
2010: Getting hired

- 22 December 2009: Google phone interview
- 15 January 2010: Presented thesis
- Signed up for ENGA04
- 20 January: On-site interview in London
- 4 February: Google offer of employment
- 29 March, start date
Terminology

Chromium + Branding = Chrome

C++ Programming Language + Clang
Starter Project

- Various bug fixes in Chrome
- Implement DeviceOrientation events

Author: hans@chromium.org
Date: Wed Aug 11 14:42:53 2010 UTC (8 years, 1 month ago)
Changed paths: 18
Log Message:

Chromium plumbing for Device Orientation.

Add the plumbing needed for communicating with the Device Orientation code in WebKit.

RenderView provides an implementation of WebKit::WebKitDeviceOrientationClient: DeviceOrientationDispatcher. This communicates with the browser-side class device_orientation::DispatcherHost.

device_orientation::Provider, responsible for providing the orientation data, is just an empty shell for now.

BUG=44654
TEST=chrome_tests --gtest_filter=DeviceOrientationBrowserTest.BasicTest

Review URL: http://codereview.chromium.org/2858049
Hi,

I saw that you worked on LLVM.

I've been (slowly, as a 20% project) trying to get Chrome to build under Clang. It's mostly been a process of reducing compiler bugs to test cases, but recently (last week) I got most of the main source tree to successfully syntax-check!

I hope to use this eventually so I can write static analysis tools for Chrome. My work in progress patch (it gets larger and smaller as I commit pieces of it) is here:

http://codereview.chromium.org/522020/show
20% Work: Clang

- Clang was very new, we were curious
- I was excited to work on something compiler related
- Developers were very excited about better diagnostics
- It was fast
-Designed for hackability
Diagnostics

```c
int f(int x) {
    int s = 0
    for (int i = 0; i < x; ++i)
        s += i;
    return s;
}
```

a.cc: In function 'int f(int)'
a.cc:3:9: error: expected ',' or ';' before 'for'
a.cc:3:25: error: 'i' was not declared in this scope
a.cc:3:35: error: expected ';' before ')' token
```c
int f(int x) {
    int s = 0
    for (int i = 0; i < x; ++i)
        s += i;
    return s;
}
```

```
a.cc:2:18: error: expected ‘;’ at end of declaration
int s = 0
```
Build Speed
Competition is good

- GCC’s diagnostics have improved a lot since then
- Build speed is more similar
In file included from a.cc:1:
./a.h:8:3: **warning:** [chromium-style] Overriding method must have "virtual" keyword.
    void foo();

1 warning generated.
What did we have to do?

- Fix many C++ errors in Chromium
- Fix many bugs found by Clang’s warnings
- File bugs for Clang
- Fix some ourselves

- Dec 2009: First Chromium patch mentioning Clang
- Sep 2010: Linux and Mac builds work
Results

- Continuous integration with Clang on all platforms (*)
- Many developers use Clang locally
- Chrome 15 for Mac built with Clang (Oct 2011)

* except Windows
trunk LLVM

Continuous integration

1-4 weeks

clang-247874-1.tgz

1 day

Chrome Canary
trunk LLVM -> clang-247874-1.tgz -> Chrome

1-4 weeks

~12 weeks
## Windows

**Issue 82385**  
Starred by 49 users

<table>
<thead>
<tr>
<th>Status:</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner:</td>
<td><a href="mailto:thakis@chromium.org">thakis@chromium.org</a></td>
</tr>
<tr>
<td>Closed:</td>
<td>Mar 21</td>
</tr>
</tbody>
</table>
| Cc:      | mbonadel@chromium.org  
kcc@chromium.org |

**Deploy Clang on windows**  
Reported by thakis@chromium.org, May 12 2011

clang's `-fms-extensions` support has improved dramatically. We should look into how viable building chrome on windows is with clang.

Showing comments 751 - 850 of 850  
[Older >](#)
Windows

Lots of good reasons

- Chrome’s largest desktop platform
- Running into limitations of MS Visual C++ compiler and linker
- Want to benefit from our compiler work on all platforms
- New tech: AddressSanitizer, CFI, ThinTLO, ...
- Crazy ideas: cross compilation, ...
How hard could it be?
Windows Support Requirements

- Want to compile Chromium w/ Clang on Windows
- Must support compiling MS system headers
- Must be binary compatible, able to link against system libraries
- Binaries must work with existing debugging, profiling, etc. tools
- Build time, binary size and run-time performance must be on par or better
- IDE integration
- Build system integration
What about MinGW?

- Minimalist GNU for Windows (MinGW)
- Allows compiling Windows programs with GCC
- Not source compatible (has its own headers)
- Not binary compatible (can’t link against MSVC-built binaries)
Command-line compatibility
Command-line compatibility

C:\src\tmp>clang-cl /O2 /Tp a.cc
C:\src\tmp>
Command-line compatibility
Command-line compatibility

C:\src\tmp>clang-cl /?
OVERVIEW: clang LLVM compiler

USAGE: clang-cl.exe [options] <inputs>

CL.EXE COMPATIBILITY OPTIONS:
  /* */   Display available options
  /arch:<value>   Set architecture for code generation
  /Bprepro   Emit an object file which cannot be reproduced over time
  /Bprepro   Emit an object file which can be reproduced over time
  /c        Don’t discard comments when preprocessing
  /c        Compile only
  /d1PP     Retain macro definitions in /E mode
  /dreportAllClassLayout Dump record layout information
  /diagnostics:caret Enable caret and column diagnostics (on by default)
  /diagnostics:classic Disable column and caret diagnostics
  /diagnostics:column Disable caret diagnostics but keep column info
  /D <macro[-value]> Define macro
  /E        Runtime encoding, supports only UTF-8
  /E        Preprocess to stdout
  /E        Preprocess to stdout
  /Fallback Fall back to cl.exe if clang-cl fails to compile
  /Fa       Output assembly code file during compilation
  /Fa<file or directory> Output assembly code to this file during compilation (with /Fa)
  /Fe<file or directory> Set output executable file or directory (ends In / or \)
  /FI <value> Include file before parsing
  /Fix<file> Set preprocessed output file name (with /P)
Source Compatibility: Hyrum’s Law

With a sufficient number of users of an API [or compiler],
it does not matter what you promise in the contract:
all observable behaviors of your system
will be depended on by somebody.

(www.hyrumslaw.com)
Source Compatibility: Preprocessor Quirks

REM This is a comment :-]
Source Compatibility: Preprocessor Quirks

#define REM / ## /

REM This is a comment :-)
Source Compatibility: Two-Phase Lookup

template<int N> int f() { return N + a; }

int a;

void g() { f<4>(); }
Source Compatibility: Two-Phase Lookup

template<typename T, typename S = Foo> class Class;

class Foo {};

template<typename T, typename S> class Class {};

warning: using the undeclared type 'Foo' as a default template argument is a Microsoft extension [-Wmicrosoft-template]
It’s the little differences: Signed enums

```c++
enum Color { RED, BLACK };

class Node {
    Color color : 1;
};
```

Enum variables are signed on Windows

This is extra surprising in bit-fields

Don’t use enum for bitfields

Google
Platform-specific features: dllexport

When building a DLL:

```c
int __declspec(dllexport) foo() { return 42; }
```

When linking against a DLL:

```c
int __declspec(dllimport) foo() { return 42; }
```
Platform-specific features: dllexport

struct __declspec(dllexport) Class {
    int foo() { return 42; }
};
Platform-specific features: dllexport

template <typename T> Base {
    int bar() { return 42; }
};

struct __declspec(dllexport) Class : public Base<int> {
    int foo() { return 42; }
};

Google
Binary Compatibility

- Application Binary Interface (ABI)
- Defines how pieces of code interact at the binary level
- For non-Windows this is mostly well documented for C++
- For Windows it is not.
ABI basics: sizes, etc.

- `long` is always 32 bits on Windows
- `long` is 32 or 64 bits on Mac/Linux on x86/x86_64
- ...

ABI: Name Mangling

Symbols are linked together by name

```c
int foo() { return 42; }
```

In C, this symbol will be called “foo” in the object file. (_foo on Windows)

In C++ it will be “_Z3foov” (Mac/Linux/...) or “?foo@@YAHXZ” (Windows)
ABI: Name Mangling

- Linux, Mac: Itanium C++ ABI section 5.1
- Windows: look at compiler output and figure it out
ABI: Name Mangling, Why?

In C++ many functions can have the same name:

```cpp
int foo(int);

int foo(double);

namespace ns { int foo(); }

class C { int foo(); };
```
ABI: Name Mangling, Why?

In C++ many functions can have the same name:

```cpp
int foo(int); // ?foo@@YAHH@Z
int foo(double); // ?foo@@YAHN@Z
namespace ns { int foo(); } // ?foo@ns@@YAHXZ
class C { int foo(); }; // ?foo@C@@QAEHXZ
```

Microsoft refers to this as “decoration” rather than “mangling”.

Google
inline void foo(bool b) {
    if (b) {
        static int x = use(&b); // ?x@4??foo@@YAX_N@Z@4HA
    } else {
        static int x = use(&b); // ?x@6??foo@@YAX_N@Z@4HA
    }
}
inline void foo(bool b) {
    if (b) {
        static int x = use(&b);  // ?x@?4??foo@@YAX_N@Z@4HA
    }
    static int x = use(&b);  // ?x@?4??foo@@YAX_N@Z@4HA
}

a.obj : fatal error LNK1179: invalid or corrupt file

Fixed in Visual Studio 2015
ABI: Calling Conventions

```c
struct S {
    int f(int a) { return x + a; }
    int x;
};
```

32-bit Linux/Mac: this and a both on the stack, return in %EAX (classic C-style call)

Windows: this in %ECX, a on the stack, return value in %EAX (__thiscall)

32-bit Win also has __stdcall, __fastcall, __vectorcall
Record Layout

```c
struct S {
    char c;
    int i;
    unsigned x : 1;
    unsigned y : 1;
};
```

Windows:
```
0 | struct S
0 |   char c
4 |   int i
8 |   unsigned int x
8 |   unsigned int y
```

Linux:
```
0 | struct S
0 |   char c
4 |   int i
8 |   unsigned int x
8 |   unsigned int y
```
Record Layout: Inheritance

```c
struct A { int a; };  // Windows: 0 | struct C
struct B { int b; };  //    4 | struct A (base)
struct C : public A,
            public B {  //    0 | struct A (base)
    int c;        //    0 | int a
};              //    4 | struct B (base)
```

```c
// Linux:
0 | struct C
0 | struct A (base)
0 | int a
4 | struct B (base)
4 | int b
8 | int c
```
Record Layout: Mysterious Padding

```c
struct S {
    virtual void f();
    int i;
    double d;
};
```

<table>
<thead>
<tr>
<th></th>
<th>Windows:</th>
<th>Linux:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>struct S</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(S vtable ptr)</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>int i</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>double d</td>
<td>8</td>
</tr>
</tbody>
</table>
```
Virtual Functions

struct S {
  virtual void f();
};

void foo(S *s) {
  s->f();
}

Windows:
0 | struct S
0 | (S vftable pointer)
VFTable for 'S' (2 entries).
0 | S RTTI
1 | void S::f()

Linux:
0 | strut S
0 | (S vtable pointer)
Vtable for 'S' (3 entries).
0 | offset_to_top (0)
1 | S RTTI
-- (S, 0) vtable address --
2 | void S::f()
Pointers to Members

```c
struct S {
    void f();
    int x;
};

struct T { void g(); };

struct U : public S, public T { };

typedef void (U::*UMemPtr)(void);

UMemPtr p1 = &U::f; // = { &f, 0 }
UMemPtr p2 = &U::g; // = { &g, 4 }
```
struct S {
    virtual void f();
    virtual void g();
};

typedef void (S::*SMemPtr)(void);

SMemPtr p1 = &S::f; // = { 1, 0 }
SMemPtr p2 = &S::g; // = { 5, 0 }
Pointers to Virtual Member Functions (Windows)

struct S {
    virtual void f();
};

typedef void (S::*SMemPtr)(void);

SMemPtr p1 = &S::f;  // = { ??_9S$BAAE, 0 }

??_9S$BA@AE:
    ; Call 1st function in S's vftable.
    movl (%ecx), %eax
    jmp *(%eax)
And many other issues

- Object file format: ELF (Linux), Mach-O (Mac), COFF (Windows)
- Debug info format: DWARF (Linux, Mac), CodeView (Windows)
- Debug info container format: PDB
Results

February 2018

Google Chrome: 64.0.3282.186 (Official Build) (64-bit) (cohort: Stable)
Revision: 9611116ee79c63602f452e4fae2242a61cf0672d-refs/branch-heads/3282@{#694}
OS: Windows
JavaScript: V8 6.4.388.46
Compiler: clang
Results

August 2018

About Version

Google Chrome: 68.0.3440.75 (Official Build) (64-bit) (cohort: 68_75_win)
Revision: cf598d63a4f1b9e7cd14f2a8433276b196e307d-refs/branch-heads/3440@(#738)
OS: Windows
JavaScript: V8 6.8.275.24
Flash: 30.0.0.113 C:\Users\thakis\AppData\Local\Google\Chrome\User
Data\PepperFlash\30.0.0.113\pepflashplayer.dll
User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/68.0.3440.75 Safari/537.36
Executable Path: C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
Profile Path: C:\Users\thakis\AppData\Local\Google\Chrome\User
Data\Default
Linker: lld-link
Results

It’s not just us...

Firefox switching to clang-cl for Windows builds (groups.google.com)
280 points by sohkamyung 84 days ago | hide | past | web | favorite | 80 comments
Results

- Chrome is now on a completely open-source toolchain
- We can fix and improve things ourselves!
- A new alternative for the Windows community
- Also we learned a lot about C++ internals.
Lessons

● Compilers are fun
● Practice your programming skills
● Participate in the programming competition
● ...
● Be part of pushing technology forward.