



# LLVM/Clang on Windows

Hans Wennborg

`hans (at) chromium.org`

10 December 2014

# Outline

## Background

- LLVM

- Clang

- Chromium

## Windows Support

- Object File Format

- Mangling

- Record Layout

- vftables / vtables

- Member pointers

- Source-level issues

- UI Compatibility

## Summary

LLVM



# LLVM

- ▶ Libraries for program analysis, optimization, codegen
- ▶ Originally written by Chris Lattner at UIUC
- ▶ Picked up by Apple in 2005
- ▶ Sub-projects: front-end, linker, debugger, ...
- ▶ No longer an acronym
- ▶ Open-source: corporate and individual contributors

# LLVM Internal Representation (IR)

- ▶ RISC-like slightly higher level than assembly
- ▶ Typed
- ▶ Abstracts most (at least many) target details
- ▶ Static Single Assignment (SSA) form
- ▶ Serializable

## Example LLVM IR Generation Code

```
Value *n = fn->arg_begin();
Value *cmp = builder.CreateICmpSLE(n, one());
builder.CreateCondBr(cmp, thenBlock, elseBlock);

builder.SetInsertPoint(thenBlock);
builder.CreateRet(n);

builder.SetInsertPoint(elseBlock);
Value *nMinusOne = builder.CreateSub(n, one());
Value *call1 = builder.CreateCall(fn, nMinusOne);
Value *nMinusTwo = builder.CreateSub(n, two());
Value *call2 = builder.CreateCall(fn, nMinusTwo);
Value *sum = builder.CreateAdd(call1, call2);
builder.CreateRet(sum);
```

# Example LLVM IR

```
define i32 @fib(i32) {
entry:
    %1 = icmp sle i32 %0, 1
    br i1 %1, label %then, label %else

then:
    ret i32 %0

else:
    %2 = sub i32 %0, 1
    %3 = call i32 @fib(i32 %2)
    %4 = sub i32 %0, 2
    %5 = call i32 @fib(i32 %4)
    %6 = add i32 %3, %5
    ret i32 %6
}
```

# Clang

- ▶ LLVM front-end
- ▶ Compiles C, C++, Objective-C
- ▶ Open-sourced by Apple in 2007
- ▶ Self-hosting since 2010



# Clang Features

- ▶ Modern
- ▶ Fast
- ▶ Expressive Abstract Syntax Tree (AST)
- ▶ Excellent diagnostics
- ▶ Hackable

# The missing semicolon

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

# The missing semicolon (GCC "5.0")

```
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'

```
    for (int i = 0; i < x; ++i)
```

^

a.cc:3:25: error: 'i' was not declared in this scope

```
    for (int i = 0; i < x; ++i)
```

^

# The missing semicolon (MSVC)

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

```
main.cpp(3): error C2143: syntax error: missing ';' before 'for'
```

# The missing semicolon (Clang)

```
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  
            ^  
            ;
```

# The typo (Clang)

```
#include <iostream>
using namespace std;

void f() {
    dout << "Hello, world!" << endl;
}
```

```
a.cc:5:9: error: use of undeclared identifier 'dout';
      did you mean 'cout'?
    dout << "Hello, world!" << endl;
     ~~~~
    cout
```

# Chromium-style Warnings

```
struct Base {  
    virtual void f();  
};  
struct Derived : public Base {  
    void f();  
};
```

```
a.cc:5:17: warning: [chromium-style] Overriding  
    method must be marked with 'override' or 'final'.  
void f();  
    ^  
    override
```

# More Hard-Core Extensions: AddressSanitizer

- ▶ Compile-time instrumentation for memory bugs
- ▶ Directly-mapped shadow memory
- ▶ Redzones between stack objects, global objects
- ▶ Intercepts malloc/free, and stdlib functions
- ▶ Each memory access consults the shadow memory first
- ▶ Finds out-of-bounds and use-after-free bugs
- ▶ Very effective



# AddressSanitizer Example

```
void f(int *arr) {  
    arr[0] = 41;  
    arr[1] = 42;  
    arr[2] = 43;  
}
```

```
int main() {  
    int data[2];  
    f(data);  
    return 0;  
}
```

Valgrind:

```
==11282== ERROR SUMMARY: 0 errors from 0 contexts
```

# AddressSanitizer Example

```
$ clang++ -g -fsanitize=address /tmp/a.cc  
$ ./a.out
```

```
==11395==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fff4bf8a4b8  
WRITE of size 4 at 0x7fff4bf8a4b8 thread T0  
#0 0x49d449 in f(int*) /tmp/a.cc:4:9  
#1 0x49d5d1 in main /tmp/a.cc:9:9  
#2 0x7f8b9fd5e76c in __libc_start_main /build/builddd/eglibc-2.15/csu/libc-start.c:226  
#3 0x49d13c in _start (/usr/local/google/work/foocafepreso/a.out+0x49d13c)
```

```
Address 0x7fff4bf8a4b8 is located in stack of thread T0 at offset 56 in frame  
#0 0x49d4bf in main /tmp/a.cc:7
```

```
This frame has 2 object(s):
```

```
[32, 36) 'retval'
```

```
[48, 56) 'data' <== Memory access at offset 56 overflows this variable
```

# Chromium

- ▶ Open source browser project
- ▶ The basis of Google Chrome
- ▶ Pushing the web forward since 2008
- ▶ ∞ users
- ▶ 650 30-day active committers
- ▶ 12+ M lines of C++
- ▶ Win, Mac, Linux, ChromeOS, Android, iOS

# Chromium and Clang

- ▶ Started experimenting in 2010
- ▶ Driven by curiosity and developer satisfaction
- ▶ Custom warnings with a Clang plugin
- ▶ Some automated refactorings
- ▶ Switched Mac release in 2011
- ▶ Heavy use of AddressSanitizer et al.
- ▶ Switched Linux release in 2014
- ▶ Most users still on Windows . . .

# Windows Support

- ▶ Still good old Intel x86
- ▶ How hard could it be?

# Object File Format

- ▶ Linux: ELF
- ▶ Mac: Mach-O
- ▶ Windows: PE/COFF
- ▶ Good news: documented, Clang support exists.

# Inside the Object File

```
_Z3fibi:                                ?fib@@YAHH@Z
    pushl   %edi
    pushl   %esi
    pushl   %eax
    movl    16(%esp), %esi
    cmpl    $2, %esi
    jge     .LBB0_1
    movl    %esi, %eax
    jmp     .LBB0_3

.LBB0_1:
    leal   -2(%esi), %eax
    movl   %eax, (%esp)
    calll _Z3fibi
    movl   %eax, %edi
    decl   %esi
    movl   %esi, (%esp)
    calll _Z3fibi
    addl   %edi, %eax

.LBB0_3:
    addl   $4, %esp
    popl   %esi
    popl   %edi
    retl

.LBB0_1:                                LBB0_1:
    leal   -2(%esi), %eax
    movl   %eax, (%esp)
    calll "?fib@@YAHH@Z"
    movl   %eax, %edi
    decl   %esi
    movl   %esi, (%esp)
    calll "?fib@@YAHH@Z"
    addl   %edi, %eax

.LBB0_3:                                LBB0_3:
    addl   $4, %esp
    popl   %esi
    popl   %edi
    retl
```

# Name Mangling

- ▶ Linux and Mac: Itanium C++ ABI §5.1
- ▶ Windows: look at compiler output, figure it out
- ▶ Good news: Clang already had some support.



# Mangling is Complicated

```
inline void f(bool x) {
    if (x) {
        // ?i@?4??f@@YAX_N@Z@4HA
        static int i = 42;
        use(&i);
    } else {
        // ?i@?6??f@@YAX_N@Z@4HA
        static int i;
        use(&i);
    }
}
```

# Mangling is Complicated

```
inline void f(bool x) {  
    if (x) {  
        // ?i@?4??f@@YAX_N@Z@4HA  
        static int i = 42;  
        use(&i);  
    }  
    // ?i@?4??f@@YAX_N@Z@4HA  
    static int i;  
    use(&i);  
}
```

# Mangling is Complicated

```
inline void f(bool x) {
    if (x) {
        // ?i@?4??f@@YAX_N@Z@4HA
        static int i = 42;
        use(&i);
    }
    // ?i@?4??f@@YAX_N@Z@4HA
    static int i;
    use(&i);
}
```

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

# What Does ABI Compatibility Mean?

- ▶ Clang and MSVC object files functionally identical
- ▶ Identical externally visible names (mangling)
- ▶ Identical record layout
- ▶ vftables, vbttables
- ▶ Member pointers
- ▶ Linkage, dllimport/export
- ▶ Calling conventions
- ▶ ...

# Record Layout

## Basics

```
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

```
struct A {  
    int a;  
};  
  
struct B {  
    int b;  
};
```

```
struct C : public A, public B {  
    int c;  
};
```

Windows:

```
0 | struct C  
0 |   struct A (base)  
0 |     int a  
4 |   struct B (base)  
4 |     int b  
8 |     int 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

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

Windows:

```
0 | struct S  
0 | (S vftable pointer)  
8 | int i  
16 | double d
```

Linux:

```
0 | struct S  
0 | (S vtable pointer)  
4 | int i  
8 | double d
```

# Virtual Functions

```
struct S {  
    virtual void f();  
};
```

Windows:

```
0 | struct S  
0 | (S vftable pointer)
```

Vftable for 'S' (2 entries).

```
0 | S RTTI  
1 | void S::f()
```

Linux:

```
0 | struct 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()
```



# Virtual Inheritance

```
struct Top {
    int x;
};

struct Left : virtual Top {
};

struct Right : virtual Top {
};

struct Bottom : Left, Right {
};

int f(Bottom *bottom) {
    Left *left = bottom;
    return g(left);
}

int g(Left *left) {
    // How to find Top::x?
    return left->x;
}
```

Windows:

```
0 | struct Bottom
0 | struct Left (base)
0 | (Left vtable pointer)
4 | struct Right (base)
4 | (Right vtable pointer)
8 | struct Top (virtual base)
8 | int x

VTable for 'Left'
0 | 0
4 | 4

VTable for 'Left' in 'Bottom'
0 | 0
4 | 8

VTable for 'Right' in 'Bottom'
0 | 0
4 | 4
```

Linux:

```
0 | struct Bottom
0 | struct Left (primary base)
0 | (Left vtable pointer)
4 | struct Right (base)
4 | (Right vtable pointer)
8 | struct Top (virtual base)
8 | int x

Vtable for 'Bottom' (6 entries).
0 | vbase_offset (8)
1 | offset_to_top (0)
2 | Bottom RTTI
-- (Bottom, 0) vtable addr --
-- (Left, 0) vtable addr --
3 | vbase_offset (4)
4 | offset_to_top (-4)
5 | Bottom RTTI
-- (Right, 4) vtable addr --
```

# Pointers to Members

```
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 }
```

# Pointers to Virtual Members (Linux)

```
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 Members (Windows)

```
struct S {  
    virtual void f();  
};  
typedef void (S::*SMemPtr)(void);  
SMemPtr p1 = &S::f; // = { ??_9S@@$BA@AE, 0 }
```

# Pointers to Virtual Members (Windows)

```
struct S {  
    virtual void f();  
};  
typedef void (S::*SMemPtr)(void);  
SMemPtr p1 = &S::f; // = { ??_9S@@$BA@AE, 0 }
```

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

# Source-level Issues

- ▶ #pragmas
- ▶ declspec
- ▶ Intrinsic
- ▶ Two-phase name lookup
- ▶ ...

# Two-Phase Name Lookup

```
template <typename T> void f(T t) {  
    foo();    // Not OK.  
    t.foo(); // OK.  
};
```

# Two-Phase Name Lookup

```
template <typename T> void f(T t) {  
    foo();    // Not OK.  
    t.foo(); // OK.  
};
```

```
void foo();  
struct S {  
    void foo();  
};
```

```
void g(S s) {  
    f(s);  
}
```



# UI Compatibility

Demo time!

# Summary

- ▶ Clang is awesome
- ▶ Want to use it on Windows
- ▶ Plenty of interesting technical problems
- ▶ Now building large real-world C++ programs

# Links

- ▶ [www.llvm.org](http://www.llvm.org)
- ▶ [www.llvm.org/builds](http://www.llvm.org/builds)
- ▶ [www.llvm.org/devmtg/2013-11/#talk11](http://www.llvm.org/devmtg/2013-11/#talk11)
- ▶ [www.llvm.org/devmtg/2014-10/#talk15](http://www.llvm.org/devmtg/2014-10/#talk15)
- ▶ [mentorembdedded.github.io/cxx-abi/abi.html](http://mentorembdedded.github.io/cxx-abi/abi.html)