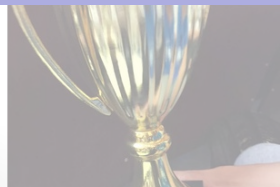
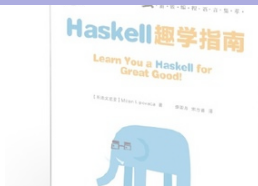


Recent work and exception handling

MaskRay

<https://maskray.me>

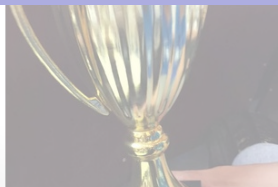
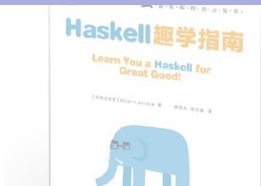
- 1 Recent work
- 2 C++ exceptions
- 3 Exception handling ABI
- 4 Personality
- 5 `.gcc_except_table`
- 6 End



MaskRay

- LLVM contributor

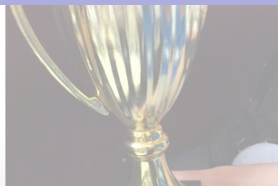
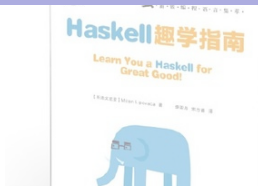




MaskRay

- LLVM contributor
- <https://github.com/MaskRay/ccls> (C++ language server)





MaskRay

- LLVM contributor
- <https://github.com/MaskRay/ccls> (C++ language server)
- Retired Algo/HPC/CTF player



Recent work (2020-08-01 to now)

- gcov (code coverage)
- LLD (linker)
- -gz, -cc1as
- integrated assembler
- debug information
- `llvm-objcopy --only-keep-debug` for Android bionic
- profile-guided optimization
- language-specific data area (LSDA)

gcov

- `-m`: demangled function names
- `--relative`: skip system headers (and other files with absolute paths)
- `--source-only` prefix
- Interaction with `-fsanitize=thread`: gcov is very early in the pipeline, Thread Sanitizer is much late.
- Optimization: Kirchhoff's circuit law
- <https://maskray.me/blog/2020-09-27-gcov-and-llvm>

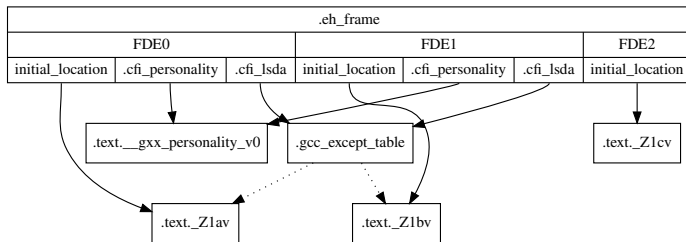
C++ exceptions

- Non-local jump to an exception handler in an ancestor call frame
- The control flow continues to unwind the stack (destructors are called for all fully-constructed non-static members and base classes)
- `std::terminate()` if no handler exists


```
void foo() { throw 0xB612; }  
void bar() { B b; foo(); }  
void qux() { try { A a; bar(); } catch (int x) {} }
```

Itanium C++ ABI: Exception Handling

- 2 parts: Level 1 Base ABI and Level 2 C++ ABI
- Base ABI: stack unwinding, common to all languages
 - `_Unwind_*` (`_Unwind_RaiseException` (two-phase process), `_Unwind_Resume`)
 - Impl: `.eh_frame_hdr`, `.eh_frame`,
`libgcc_s.so.1/libgcc_eh.a/libunwind` (`--unwindlib=libunwind`)
- C++ ABI: interoperability of C++ implementations
 - `__cxa_*` (`__cxa_allocate_exception`, `__cxa_throw`, `__cxa_begin_catch`), personality, language-specific data area
 - Impl: `.gcc_except_table`, `libsupc++`, `libc++abi`. (also)



- `.eh_frame` contains CIE (Common Information Entry) and FDE (Frame Description Entry)
- CIE references a personality routine in `.text`
- FDE references language-specific data area in `.gcc_except_table`
- Dotted edges are present only if basic block sections or RISC-V `-mrelax`, i.e. LSDA normally has no relocation

```

static _Unwind_Reason_Code unwind_phase1(unw_context_t *uc, _Unwind_Context *ctx,
                                         _Unwind_Exception *obj) {
    // Search phase: unwind and call personality with _UA_SEARCH_PHASE for each frame
    // until a handler (catch block) is found.
}

static _Unwind_Reason_Code unwind_phase2(unw_context_t *uc, _Unwind_Context *ctx,
                                         _Unwind_Exception *obj) {
    // Cleanup phase: unwind and call personality with _UA_CLEANUP_PHASE for each frame
    // until reaching the handler. Restore the register state and transfer control.
}

_Unwind_Reason_Code _Unwind_RaiseException(_Unwind_Exception *obj) {
    unw_context_t uc; __unw_getcontext(&uc);
    _Unwind_Context ctx;
    _Unwind_Reason_Code phase1 = unwind_phase1(&uc, &ctx, obj);
    if (phase1 != _URC_NO_REASON) return phase1;
    unwind_phase2(&uc, &ctx, obj);
}

void __cxa_throw(void *thrown, std::type_info *tinfo, void (*destructor)(void *)) {
    uncaughtExceptions++;
    __cxa_exception *hdr = (__cxa_exception *)thrown - 1;
    hdr->exceptionType = tinfo; hdr->destructor = destructor;
    _Unwind_RaiseException(&hdr->unwindHeader);
    // Failed to unwind, e.g. the .eh_frame FDE is absent.
    __cxa_begin_catch(&hdr->unwindHeader); std::terminate();
}

void foo() { __cxa_exception *thrown = __cxa_allocate_exception(4); *thrown = 42;
             __cxa_throw(thrown, &typeid(int), /*destructor=*/nullptr); }

void bar() { B b; qux(); return; landing_pad: b.~B(); _Unwind_Resume(); }
// void qux() { try { A a; bar(); } catch (int x) {} }
void qux() { A a; bar(); return; landing_pad: __cxa_begin_catch(obj); __cxa_end_catch(obj); }

```

Life of an exception

- `foo` allocates a `__cxa_exception` object, sets the thrown value, and calls `__cxa_throw`
- `__cxa_throw` sets fields in the header (type info, destructor, etc) and call Base ABI specified `_Unwind_RaiseException` (on error: `std::terminate()`)
- `_Unwind_RaiseException` calls `unw_phase1` and then `unw_phase2`
- `unw_phase1` locates the matching catch block
- `unw_phase2` transfers control to a cleanup handler
- The handler performs cleanup and jumps back via `_Unwind_Resume`
- `unw_phase2` eventually transfers control to the matching catch block
- The final landing pad returns the exception object via `__cxa_begin_catch`

└ Exception handling ABI

└ Level 1: Base ABI

```

// Find .eh_frame CIE/FDE associated with the IP stored in ctx.
// Get personality and LSDA from CIE and FDE.
static void unw_init_local(unw_context_t *uc, _Unwind_Context *ctx) {
    Iterate PT_GNU_EH_FRAME (.eh_frame_hdr) and find the one defining IP.
    Find the .eh_frame FDE referenced by the .eh_frame_hdr entry.
    Find the .eh_frame CIE referenced by the CIE.
    ctx->start_ip = fdeInfo.pcStart; ctx->end_ip = fdeInfo.pcEnd;
    ctx->lsda = fdeInfo.lsda; ctx->personality = cieInfo.personality;
}

// Execute .cfi_* to restore PC, SP, and other callee-saved registers in ctx
static bool step(_Unwind_Context *ctx) { ... }

static _Unwind_Reason_Code unwind_phase1(unw_context_t *uc, _Unwind_Context *ctx,
                                         _Unwind_Exception *obj) {
    unw_init_local(uc, ctx);
    for(;;) {
        if (ctx->fdeMissing) return _URC_END_OF_STACK;
        if (!step(ctx)) return _URC_FATAL_PHASE1_ERROR;
        ctx->getFdeAndCieFromIP();
        if (!ctx->personality) continue;
        switch (ctx->personality(1, _UA_SEARCH_PHASE, obj->exception_class, obj, ctx)) {
            case _URC_CONTINUE_UNWIND: break;
            case _URC_HANDLER_FOUND:
                unw_get_reg(ctx, UNW_REG_SP, &obj->private_2);
                return _URC_NO_REASON;
            default: return _URC_FATAL_PHASE1_ERROR; // e.g. stack corruption
        }
    }
    return _URC_NO_REASON;
}

```

└ Exception handling ABI

└ Level 1: Base ABI

```

static void unw_resume(_Unwind_Exception *ctx) {
    Jump to a landing pad (cleanup or the matching catch block).
    Similar to longjmp: set callee-saved registers, SP and IP.
}

static _Unwind_Reason_Code unwind_phase2(unw_context_t *uc, _Unwind_Context *ctx,
                                         _Unwind_Exception *obj) {
    unw_init_local(uc, ctx);
    for(;;) {
        if (ctx->fdeMissing) return _URC_END_OF_STACK;
        if (!step(ctx)) return _URC_FATAL_PHASE2_ERROR;
        ctx->getFdeAndCieFromIP();
        if (!ctx->personality) continue;
        _Unwind_Action action = _UA_CLEANUP_PHASE;
        size_t sp;
        unw_get_reg(ctx, UNW_REG_SP, &sp);
        if (sp == obj->private_2) action |= _UA_HANDLER_FRAME;
        switch (ctx->personality(1, action, obj->exception_class, obj, ctx)) {
            case _URC_CONTINUE_UNWIND:
                break;
            case _URC_INSTALL_CONTEXT:
                unw_resume(ctx); // Return if there is an error
                return _URC_FATAL_PHASE2_ERROR;
            default: return _URC_FATAL_PHASE2_ERROR; // Unknown result code
        }
    }
    return _URC_FATAL_PHASE2_ERROR;
}

```

Personality

- Bridge between Level 1 Base ABI and Level 2 C++ ABI
- In C++, usually `__gxx_personality_v0` (sjlj: `__gxx_personality_sj0`)
- GCC `libstdc++-v3/libsupc++/eh_personality.cc` and `libc++abi`, defined in `src/cxa_personality.cpp`
- `__gxx_personality_v0` parses the referenced `.gcc_except_table` piece, locates the call-site code range, and executes specified actions (e.g. jump to a label).
- Roughly, the function is partitioned by `try` into multiple code ranges.

.gcc_except_table

- Interpreted by `__gxx_personality_v0`
- For each code range, describe the landing pad (catch block) and actions (e.g. skip if type mismatch)
- Header + call-site records + action records
- call-site record: call site, landing pad, action record (1 indicates the start)
- action record: type filter, next action record

```

main:                                # @main
.Lfunc_begin0:
    .cfi_startproc
    .cfi_personality 3, __gxx_personality_v0
    .cfi_lsda 3, .Lexception0
# %bb.0:                              # %entry
    pushq    %rax
    .cfi_def_cfa_offset 16
.Ltmp0:
    callq    _Z2fbv                  # try region
.Ltmp1:
.LBBO_2:
    xorl     %eax, %eax
    popq     %rcx
    .cfi_def_cfa_offset 8
    retq
.LBBO_1:                              # landing pad
    .cfi_def_cfa_offset 16
.Ltmp2:
    movq     %rax, %rdi
    callq    __cxa_begin_catch
    movl     (%rax), %esi
    movl     $.L.str, %edi
    xorl     %eax, %eax
    callq    printf
    callq    __cxa_end_catch
    jmp      .LBBO_2
.Lfunc_end0:
    .size    main, .Lfunc_end0-main
    .cfi_endproc

```

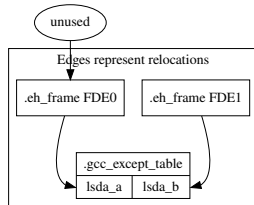
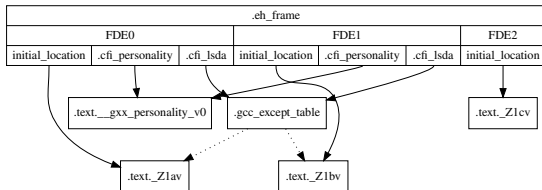
```

.section      .gcc_except_table,"a",@progbits
.p2align     2
GCC_except_table0:
.Lexception0:
.byte       255                # @LPStart Encoding = omit
.byte       3                  # @TType Encoding = udata4
.uleb128     .Lttbase0-.Lttbaseref0 # The start of action records
.Lttbaseref0:
.byte       1                  # Call site Encoding = uleb128
.uleb128     .Lcst_end0-.Lcst_begin0
.Lcst_begin0:
.uleb128     .Ltmp0-.Lfunc_begin0  # 2 call site code ranges
.uleb128     .Ltmp1-.Ltmp0          # >> Call Site 1 <<
.uleb128     .Ltmp2-.Lfunc_begin0  # Call between .Ltmp0 and .Ltmp1
.uleb128     .Ltmp2-.Lfunc_begin0  # jumps to .Ltmp2
.byte       1                    # On action: 1
.uleb128     .Ltmp1-.Lfunc_begin0  # >> Call Site 2 <<
.uleb128     .Lfunc_end0-.Ltmp1    # Call between .Ltmp1 and .Lfunc_end0
.byte       0                    # has no landing pad
.byte       0                    # On action: cleanup
.Lcst_end0:
.byte       1                    # >> Action Record 1 <<
                                # Catch TypeInfo 1
                                # No further actions
.byte       0
.p2align     2
                                # >> Catch TypeInfos <<
                                # TypeInfo 1
.long        _ZTIi
.Lttbase0:

```

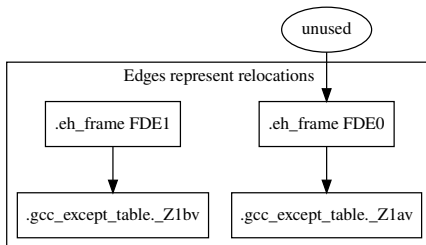
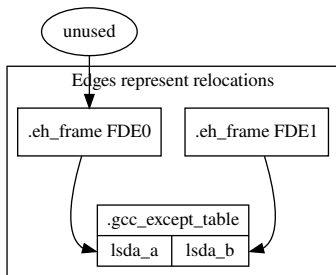
Monolithic .gcc_except_table

- As of Clang 11, there is one monolithic .gcc_except_table
- Linker `--gc-sections`: input sections are atoms
- Unused .gcc_except_table (the "referenced" .text sections are discarded) cannot be discarded



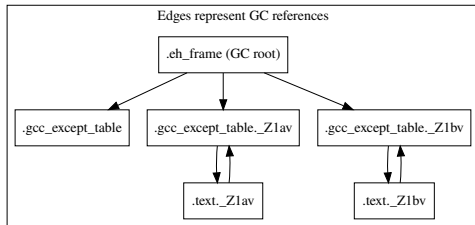
Fragmented .gcc_except_table

- <https://reviews.llvm.org/D83655>: Split up .gcc_except_table



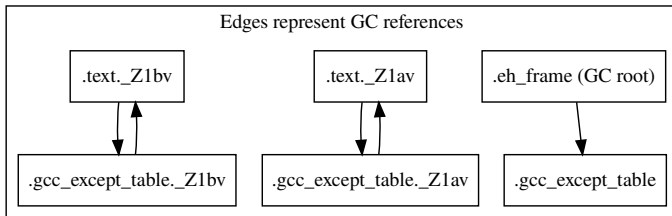
Is that so simple?

- No! `.text.*` in COMDAT groups cannot be GCed with fragmented `.gcc_except_table.*` (code size increase)
- LLD handles `--gc-sections` before GCing `.eh_frame`
- During GC, all pieces in `.eh_frame` are live (GC root). They mark all `.gcc_except_table.*` live
- A `.gcc_except_table.*` marks other members (`.text.*`) in the same group live (linker rule)



Let's fix LLD!

- <https://reviews.llvm.org/D91579>: for .eh_frame, don't mark .gcc_except_table within a COMDAT group



Future work

- Clang `.gcc_except_table` is inefficient for pass-through purposes. GCC produces header-only LSDA (4 bytes).
- Clang/LLD interop: garbage collect unused `.gcc_except_table` not within COMDAT groups
- Efficient (space/performance) `.eh_frame` (very difficult; (current) compact unwinding has lots of limitations;
<https://maskray.me/blog/2020-11-08-stack-unwinding>)