

Signal Exceptions

Parte 1



Señales

Signal target:
"yes"

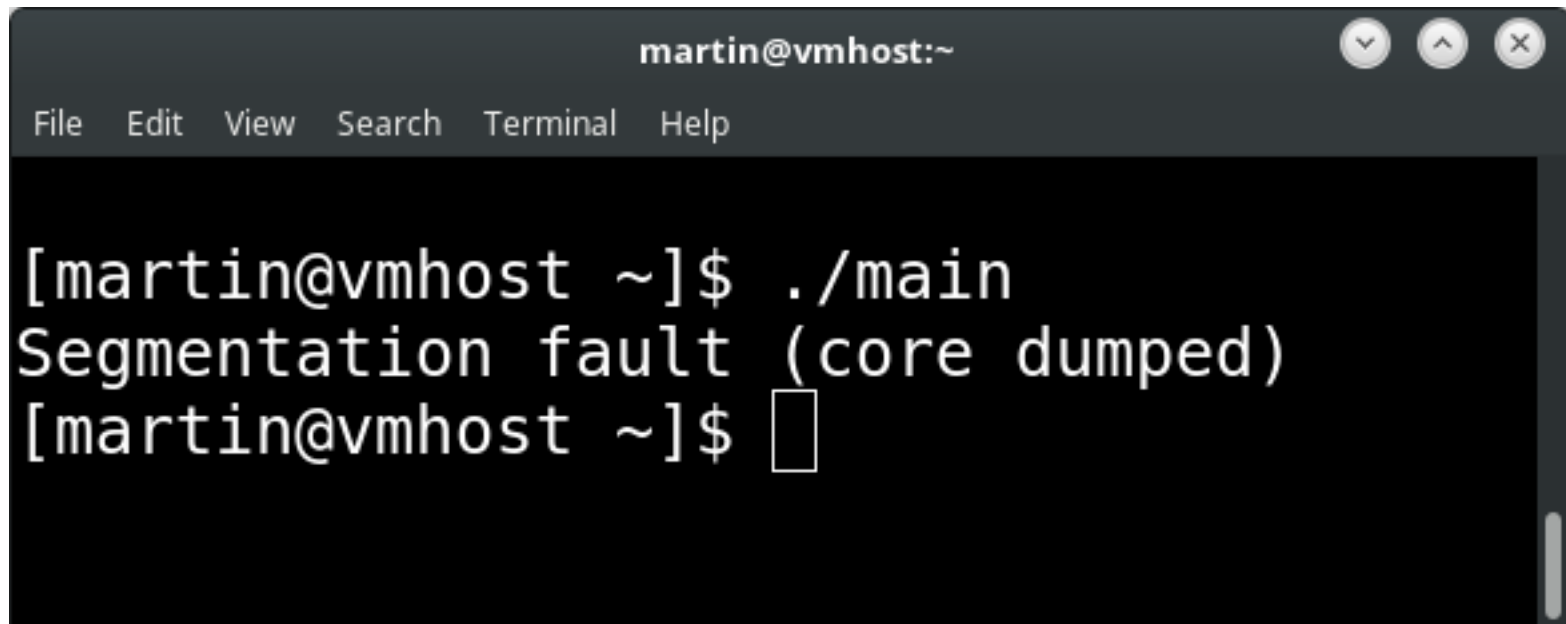
Signal:
0x2 (SIGINT)

Backtrace:

```
#0  send_signal at kernel/signal.c
...
#8  n_tty_receive_char_special (c=3) at
drivers/tty/n_tty.c
...
#15 receive_buf at drivers/tty/tty_buffer.c
...
#19 kthread at kernel/kthread.c
...
```

¿Qué pasó acá?

```
int main(void) {  
    long addr = 0x0L;  
    *(int*)addr = 0;  
}
```

A terminal window titled "martin@vmhost:~" with a menu bar containing "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows the command `./main` being executed, which results in a "Segmentation fault (core dumped)" message. The prompt `[martin@vmhost ~]$` is shown again with a cursor.

```
martin@vmhost:~  
File Edit View Search Terminal Help  
[martin@vmhost ~]$ ./main  
Segmentation fault (core dumped)  
[martin@vmhost ~]$
```

Señales

Signal target:

"main"

Signal:

0xb (SIGSEGV)

Backtrace:

#0 **send_signal** at kernel/signal.c

...

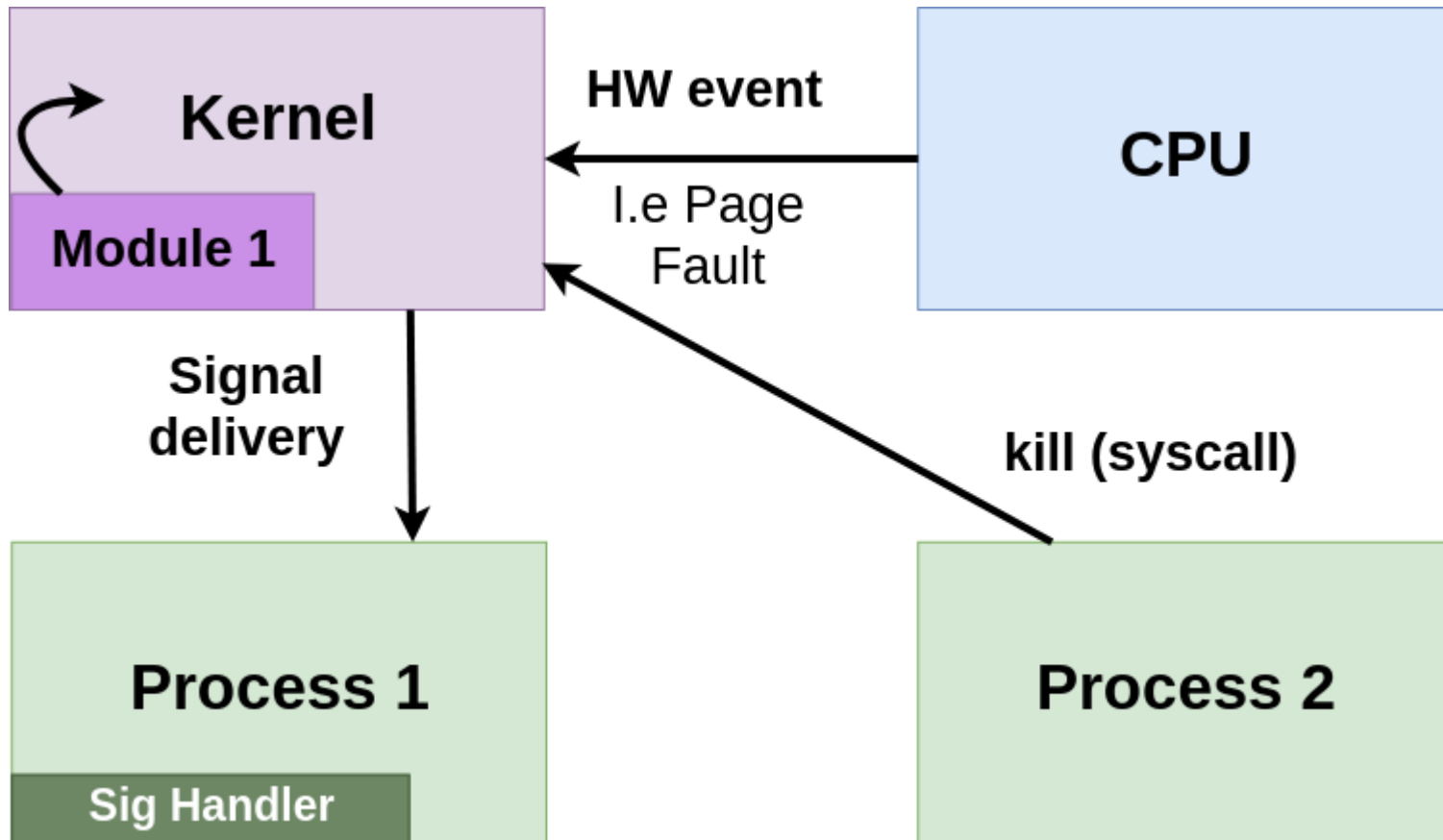
#6 **bad_area** (address=0) at arch/x86/mm/fault.c

...

#10 **async_page_fault** at
arch/x86/entry/entry_64.S

...

Señales



Señales

- Portable Operating System Interface (**POSIX**)
 - APIs, shells, utilidades, etc.
 - Distros de Linux siguen la mayor parte del estándar
 - Señales: mecanismo asincrónico de **IPC** para notificar procesos o threads
 - Originadas en Bell Labs Unix (1970s)
 - SIGSEGV, SIGKILL, SIGILL, SIGFPE, etc.
 - Handlers por defecto o personalizados



Señales



- Instalación de un handler de señal (API)

```
void sigsegv_handler (int signal,
                      siginfo_t* info,
                      void* context) {
    ...
}

void main() {
    struct sigaction act = {0x0};
    act.sa_sigaction = sigsegv_handler;
    act.sa_flags = SA_SIGINFO;
    sigaction(SIGSEGV, &act, NULL);
}
```

Señales

- El handler es global para el grupo de procesos
- Múltiples libs, ¿un solo handler?
 - Demultiplexación de señales
- El handler se ejecuta en el contexto de un trap
 - Solo se pueden invocar funciones *async-signal-safe*
 - Reentrancia: ¿qué sucede si una señal interrumpe la ejecución de una función no-reentrante y desde el handler se la invoca nuevamente?
 - Handler reentrante respecto a variables globales

Idea







```
void main(void) {  
    __try {  
        long addr = 0x0L;  
        *(int*)addr = 0;  
    } __catch {  
        printf("SIGSEGV\n");  
    }  
}
```

```
gcc -fsignal-exceptions -o main main.c
```

Alcance



- GCC 
- GNU Linker 
- Glibc 
- Kernel 

Excepciones en C++



- ¿Qué es una excepción?

```
void f(void) {  
    throw std::exception();  
}  
  
int main(void) {  
    try {  
        f();  
    } catch (std::exception& e) {  
        std::cout << "Catch" <<  
            std::endl;  
    }  
}
```

Compilador y Runtime

00000000004011d2 <_Z1fv>:

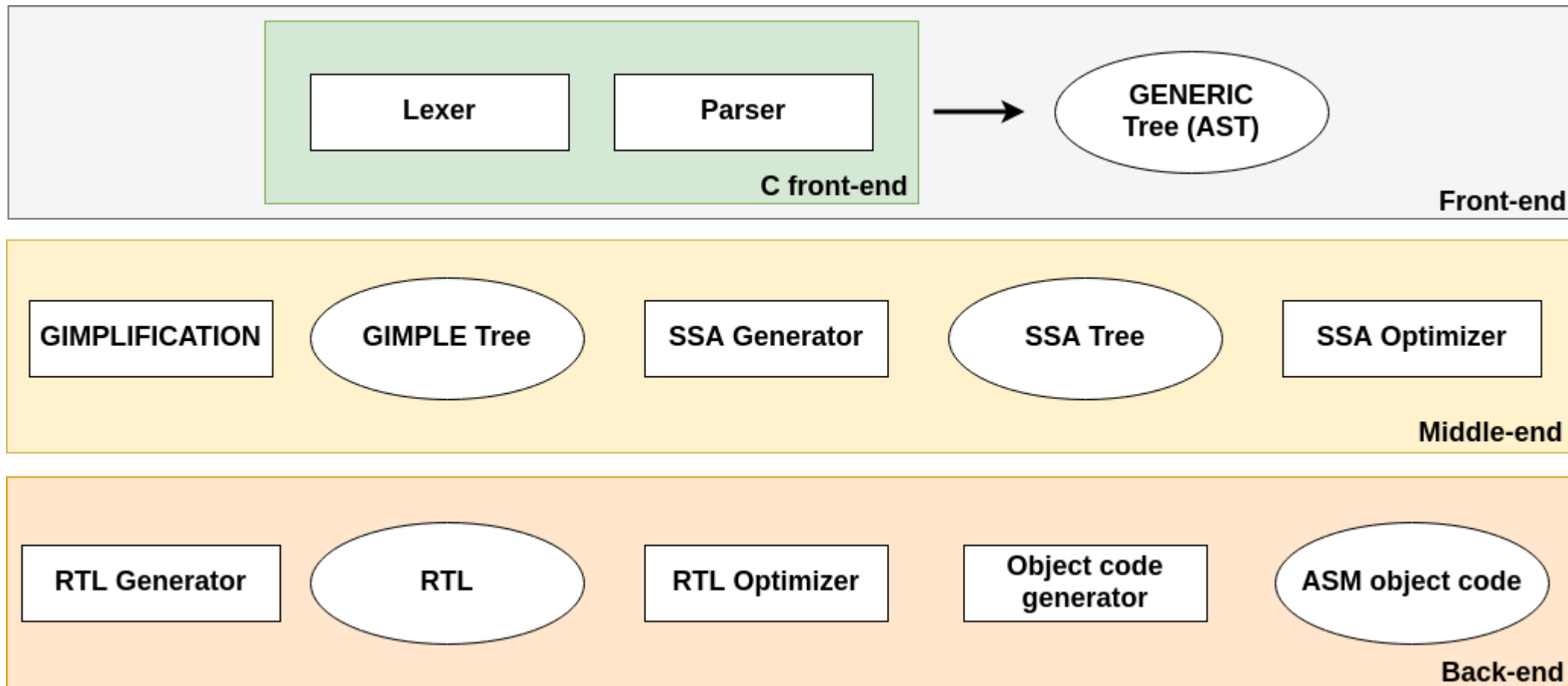
```
4011e0: e8 8b fe ff ff      callq 401070 <__cxa_allocate_exception@plt>
4011e5: 48 89 c3            mov  %rax,%rbx
4011e8: 48 89 df            mov  %rbx,%rdi
4011eb: e8 d6 00 00 00     callq 4012c6 <_ZNSt9exceptionC1Ev>
4011f0: ba f0 10 40 00     mov  $0x4010f0,%edx
4011f5: be c0 3d 40 00     mov  $0x403dc0,%esi
4011fa: 48 89 df            mov  %rbx,%rdi
4011fd: e8 7e fe ff ff      callq 401080 <__cxa_throw@plt>
```

libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007fca1cd66000)

8f4b0 FUNC GLOBAL DEFAULT **__cxa_throw@@CXXABI_1.3**

- GCC
- libgcc (libgcc_s.so.1)
 - Runtime con stack unwinder (setjmp/longjmp en C)

GCC overview



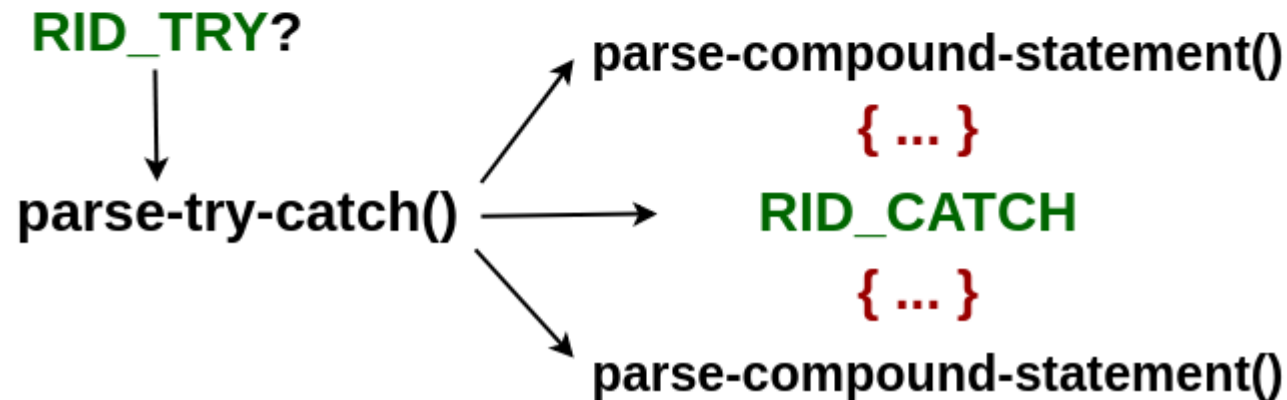
Análisis léxico y parsing



C - Tokenizer

`__try` → `RID_TRY`
`__catch` → `RID_CATCH`

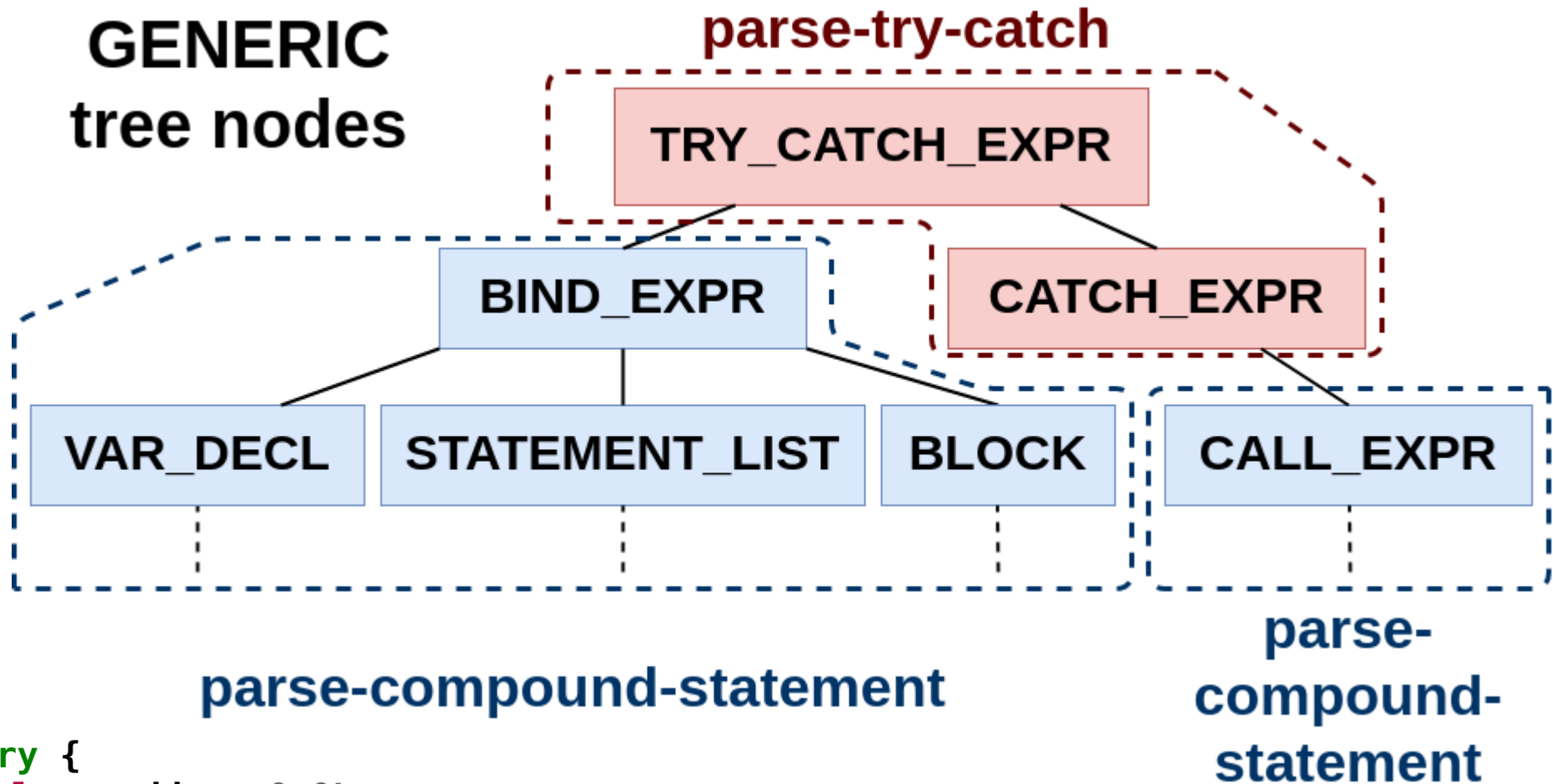
C - Parser



Abstract Syntax Tree (AST)



GENERIC
tree nodes



```
__try {  
    long addr = 0x0L;  
    *(int*)addr = 0;  
} __catch {  
    printf("SIGSEGV\n");  
}
```

Optimizaciones



- Remover código muerto y anotaciones: ¿puede una instrucción hacer throw?
- Para SIGSEGV son relevantes los accesos de lectura o escritura a memoria
 - ¿Todos?
 - Referencias a variables
 - Referencias a memoria (ej.: *(expr))
 - Arrays con índices seguros (conocidos en tiempo de compilación)
 - Inline assembly

Output #1: .text (código)



0000000000401132 <main>:

```
401132: 55          push  %rbp
401133: 48 89 e5    mov   %rsp,%rbp
401136: 48 83 ec 10 sub   $0x10,%rsp
```

```
40113a: 48 c7 45 f8 00 00 00 00  movq  $0x0,-0x8(%rbp)
401142: 48 8b 45 f8    mov   -0x8(%rbp),%rax
401146: c7 00 00 00 00 00    movl  $0x0,(%rax)
```

}
__try
}

```
40114c: b8 00 00 00 00    mov   $0x0,%eax
401151: eb 0c          jmp  40115f <main+0x2d>
```

```
401153: bf 04 20 40 00    mov   $0x402004,%edi
401158: e8 e3 fe ff ff    callq 401040 <puts@plt>
40115d: eb ed          jmp  40114c <main+0x1a>
```

}
__catch
}

```
40115f: c9          leaveq
401160: c3          retq
```

Output #2: .gcc_except_table



0xFF ----- formato de los landing pads:
DW_EH_PE_omit

0x03 ----- formato de la Tabla de Tipos: udata4

0x11 (17) ----- offset al final de la Tabla de Tipos

0x01 ----- formato de los offset en las regiones: uleb128

0x08 ----- largo de la Tabla de Call-sites (regiones) -----

0x14 (20) ----- región 0: comienzo = offset 20 -----

0x06 ----- región 0: largo = 6 bytes -----

0x21 (33) ----- región 0: landing pad = offset 33 -----

0x01 ----- región 0: acción = 1 -----

0x26 (38) ----- región 1: comienzo = offset 38 -----

0x07 ----- región 1: largo = 7 bytes -----

0x00 ----- región 1: landing pad = 0 -----

0x00 ----- región 1: acción = 0 -----

0x01 ----- Action Record Table Entrada #1 - filter -----

0x00 ----- Action Record Table Entrada #1 - next -----

0x00 ----- Padding bytes para alineación -----

0x00000000 - Tabla de Tipos

Output #2: .gcc_except_table



0000000000401132 <main>:

401132:	55	push	%rbp	- - -	
401133:	48 89 e5	mov	%rsp,%rbp	- - -	
401136:	48 83 ec 10	sub	\$0x10,%rsp	- - -	no-throw
40113a:	48 c7 45 f8 00 00 00 00	movq	\$0x0,-0x8(%rbp)	- - -	
401142:	48 8b 45 f8	mov	-0x8(%rbp),%rax	- - -	
401146:	c7 00 00 00 00 00	movl	\$0x0,(%rax)	- - -	Región 0
40114c:	b8 00 00 00 00	mov	\$0x0,%eax	- - -	no-throw
401151:	eb 0c	jmp	40115f <main+0x2d>	- - -	
401153:	bf 04 20 40 00	mov	\$0x402004,%edi	- - -	Región 0: Landing Pad
401158:	e8 e3 fe ff ff	callq	401040 <puts@plt>	- - -	
40115d:	eb ed	jmp	40114c <main+0x1a>	- - -	Región 1
40115f:	c9	leaveq			
401160:	c3	retq			

Output #3:

`.eh_frame & .eh_frame_hdr`



- Información para hacer unwinding del call stack
- Call Frame Information (CFI) #1
 - Common Information Entry (CIE)
 - Frame Description Entry (FDE) #1
 - Call Frame Instructions
 - ...
- ...

¿Y entonces?



- Al ejecutarse un binario con información de excepciones:

1) ¿**Hay que** registrar un handler de señales?

2) ¿**Quién y cuándo** lo registra?

3) ¿**Qué sucede** si es llamado en algún momento?

Registro del handler

- Los binarios generados por GCC con manejo de excepciones son indistinguibles
 - Ej.: ¿excepciones setjmp/longjmp o signal-exceptions?
- Idea: generar artificialmente desde GCC una llamada al runtime en los binarios con signal-exceptions
 - Solo una llamada (tiempo de carga)
 - Si el runtime no tiene registrado el handler, lo registra

Registro del handler

```
__attribute__((constructor,weak))  
void __init_signal_exceptions (void) {  
    __builtin_register_signal_exceptions();  
}
```

```
401161: 55          push  %rbp  
401162: 48 89 e5    mov   %rsp,%rbp  
401165: e8 c6 fe ff callq 401030 <_register_signal_exceptions@plt>  
40116a: 5d          pop   %rbp  
40116b: c3          retq
```

Hex dump of section '.init_array':

```
0x00403dd8 30114000 00000000 61114000 00000000
```

Binario compilado

libgcc (runtime)

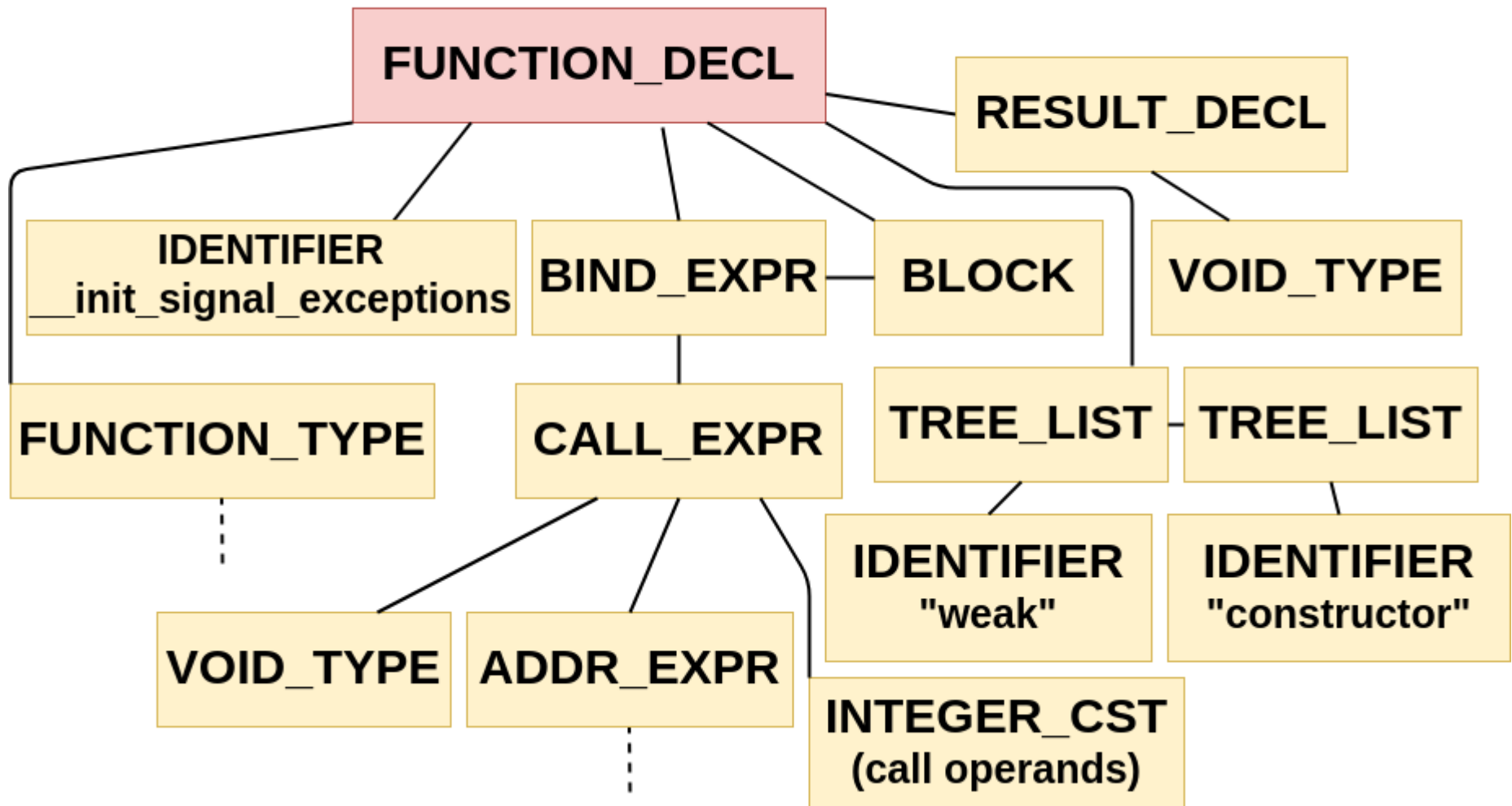
readelf -s libgcc_s.so.1

```
011000 164 FUNC GLOBAL DEFAULT _register_signal_exceptio
```

Inyección de nodos en el AST



`cgraph_node::add_new_function (...)`



Stack unwinding (libgcc)

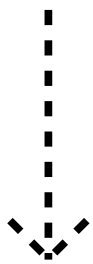


```
void _register_signal_exceptions (void) {  
    ...  
  
    if (sigexcept_status != SIGEXCEPT_UNREGISTERED)  
        return;  
    ...  
  
    if (sigaction(SIGSEGV, &sigsegv_sa, NULL) != 0)  
        goto error;  
  
    sigexcept_status = SIGEXCEPT_REGISTERED;  
  
    return;  
    ...  
}
```

Stack unwinding (libgcc)



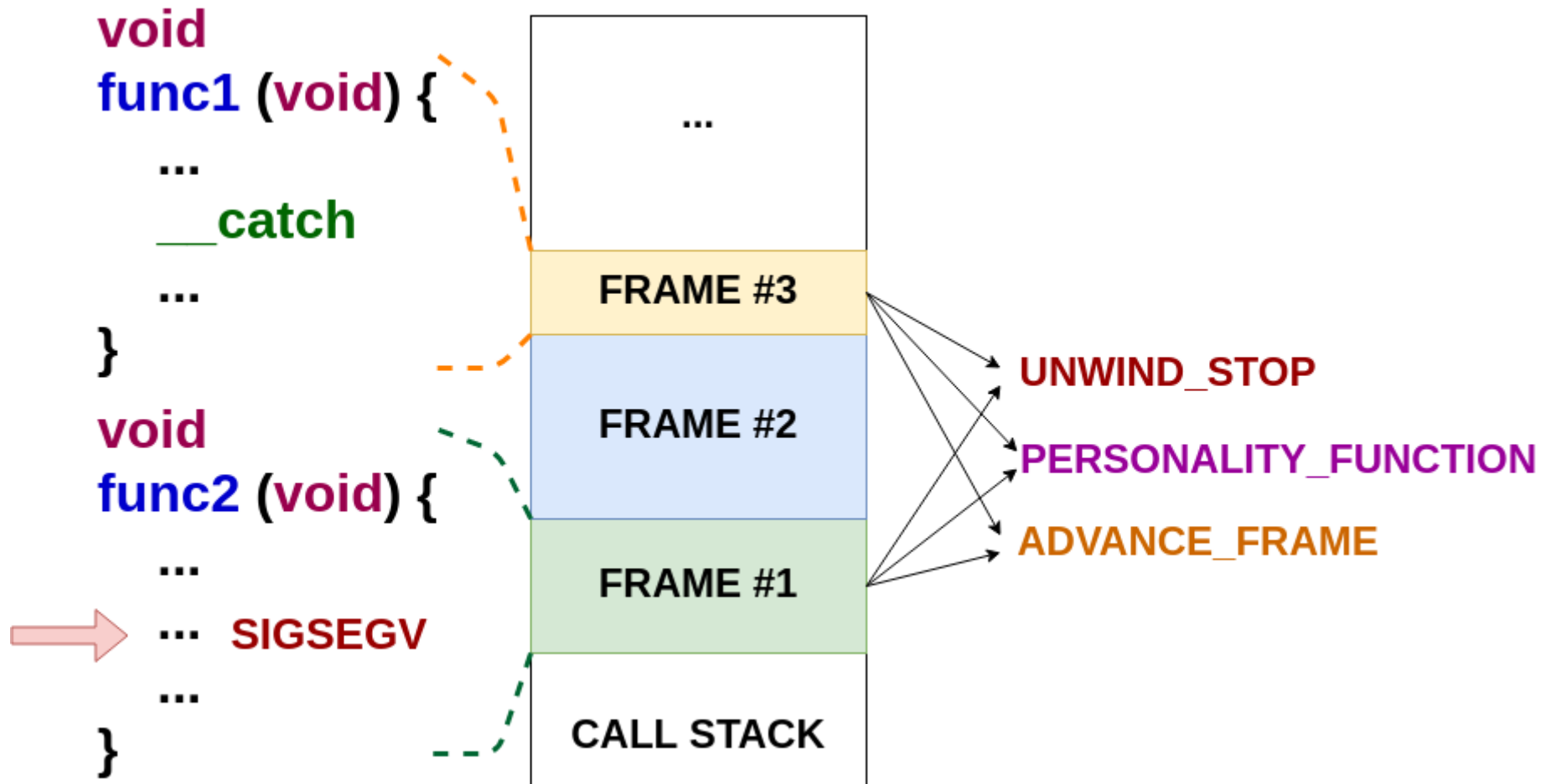
```
static void x86_64_sigexcept_sigsegv_handler (  
    int signal, siginfo_t* info, void* context) {  
    context.gregs[RSP] -= sizeof(void*);  
    *(context.gregs[RSP]) = context.gregs[RIP] + 1;  
    context.gregs[RIP] = sigexcept_unwind_trampoline;  
}
```

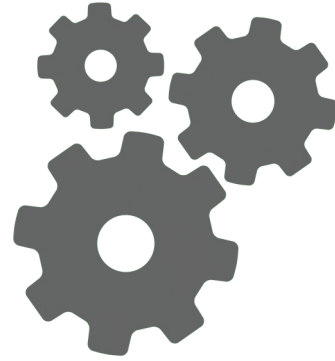


```
callq 401080 <__cxa_throw@plt>  
xyz ... C++
```

```
static void sigexcept_unwind_trampoline (void) {  
    struct _Unwind_Exception *exc = ...;  
    exc->exception_cleanup = sigexcept_exception_cleanup;  
    _Unwind_ForcedUnwind(exc, sigexcept_unwind_stop, NULL);  
}
```

Stack unwinding (libgcc)





Demo

Trabajo futuro



```
void main(void) {  
    __try {  
        long addr = 0x0L;  
        *(int*)addr = 0;  
    } __catch (SIGSEGV : int signal,  
siginfo_t* info, void* context) {  
        printf("SIGSEGV\n");  
    }  
}
```

gcc -fsignal-exceptions -o main main.c



¡Gracias!

<https://martin.uy/blog/gcc-signal-exceptions-part-1>

Créditos

- Light bulb icon: <https://www.flaticon.com/authors/freepik>
- Tick and cross icons: <https://www.iconfinder.com/iconpack>
- Finish flag icon: <https://www.flaticon.com/authors/smashicons>
- Warning icon:
<http://www.iconarchive.com/show/noto-emoji-symbols-icons-by-google/73028-warning-icon.html>
- Owl and rocket icons: <http://www.iconarchive.com/artist/thesquid.ink.html>