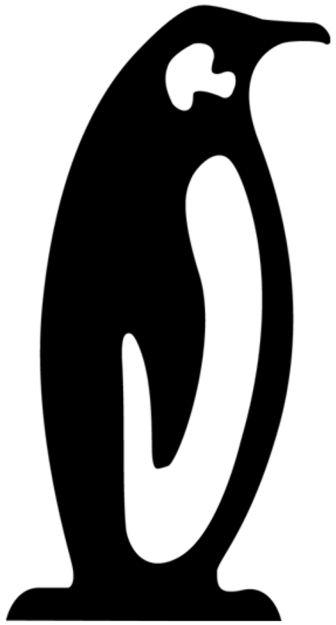


Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Integral Atomic Stack Switching for IST Exceptions

Jiangshan Lai (Ant Group)

Agenda

X86 IST
Motivation
Atomic-IST-entry
RETBLEED

X86 IST

Linus Torvalds [link]

I absolutely despise all the x86 "indirect system structures". They are horrible garbage. IST is only yet another example of that kind of brokenness, and annoys me particularly because it (and swapgs) were actually making x86 _worse_.

Thomas Gleixner: [link]

It's a sad state of affairs that I have to write this mail at all and it's nothing else than an act of desperation.

The x86 exception handling including the various ways of syscall entry/exit are a constant source of trouble. Aside of being a functional disaster quite some of these issues have severe security implications.

SYSCALL GAP

SYSCALL/SYSRET
instructions do not switch the
STACK nor the GSBASE

Current approaches

```
* +-----------------------------------------------------+
* | original SS                                         |
* | original Return RSP                                 |
* | original RFLAGS                                     |
* | original CS                                         |
* | original RIP                                        |
* +-----------------------------------------------------+
* | temp storage for rdx                                |
* +-----------------------------------------------------+
* | "NMI executing" variable                            |
* +-----------------------------------------------------+
* | iret SS          } Copied from "outermost" frame    |
* | iret Return RSP  } on each loop iteration; overwritten |
* | iret RFLAGS      } by a nested NMI to force another |
* | iret CS          } iteration if needed.             |
* | iret RIP         }                                  |
* +-----------------------------------------------------+
* | outermost SS          } initialized in first_nmi;   |
* | outermost Return RSP  } will not be changed before  |
* | outermost RFLAGS      } NMI processing is done.     |
* | outermost CS          } Copied to "iret" frame on each |
* | outermost RIP         } iteration.                  |
* +-----------------------------------------------------+
* | pt_regs                                             |
* +-----------------------------------------------------+
```

```
/*
 * Switch off the IST stack to make it free for nested exceptions. The
 * vc_switch_off_ist() function will switch back to the interrupted
 * stack if it is safe to do so. If not it switches to the VC fall-back
 * stack.
 */
movq    %rsp, %rdi              /* pt_regs pointer */
call    vc_switch_off_ist
movq    %rax, %rsp              /* Switch to new stack */
```

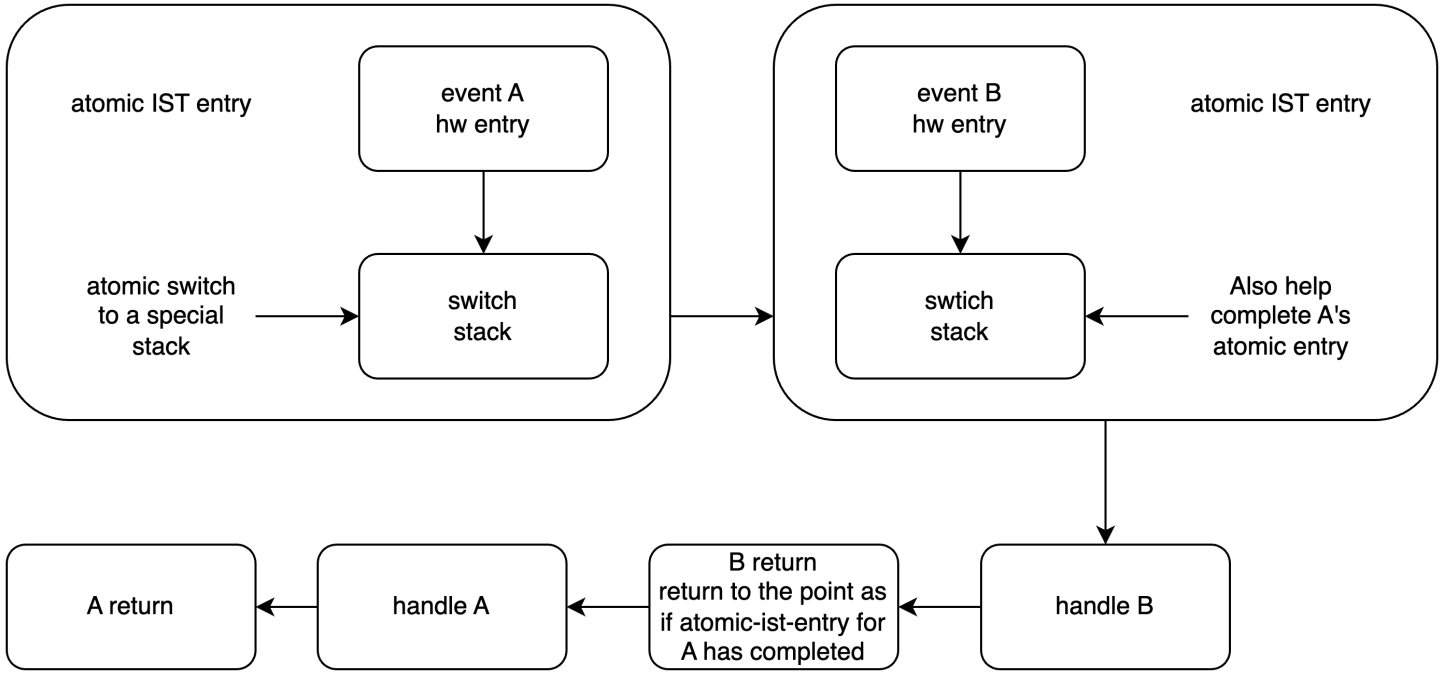
```
/*
 * Reserve additional 8 bytes and store old IST value so this
 * adjustment can be unrolled in __sev_es_ist_exit().
 */
new_ist -= sizeof(old_ist);
*(unsigned long *)new_ist = old_ist;

/* Set new IST entry */
this_cpu_write(cpu_tss_rw.x86_tss.ist[IST_INDEX_VC], new_ist);
```

Motivation

- Convert ASM entry code to C code
- Integral Entry with PVM switcher
- Address Space Isolation

Atomic-IST-entry



Abortable, Idempotently-Replicable

```
static __always_inline void pti_switch_to_kernel_cr3(unsigned long user_cr3)
{
        /*
         * Clear user PAGE_TABLE_ISOLATION and PCID mask, point CR3
         * at kernel pagetables:
         */
        unsigned long cr3 = user_cr3 & ~PTI_USER_PGTABLE_AND_PCID_MASK;

        if (static_cpu_has(X86_FEATURE_PCID))
                cr3 |= X86_CR3_PCID_NOFLUSH;

        native_write_cr3(cr3);
}

static __always_inline void ist_switch_to_kernel_cr3(unsigned long saved_cr3)
{
        if (static_cpu_has(X86_FEATURE_PTI)) {
                if (saved_cr3 & PTI_USER_PGTABLE_MASK)
                        pti_switch_to_kernel_cr3(saved_cr3);
        }
}
```

airworthiness certificated

How to make an Abortable, Idempotently-Replicable Atomic-IST-entry

Just saving the interrupted context is all we need for an entry before handling the event

Save

As an atomic entry, the original IST must be freed for next event entry, introduce an IST mian stack, copy and switch to it

Save-Copy-Switch

As an abortable, atomic procedure, it should has a finish line to commit, use the fininal instruction switching the %rsp onto the main stack as the commit instruction

Save-Copy-Commit(switch)

The save sub procedure is not hardware-atomically, which can be interrupted and abort at any point, split the save procedure as two procedures

Save(head)-Save(gp regs)-Copy-Commit(switch)

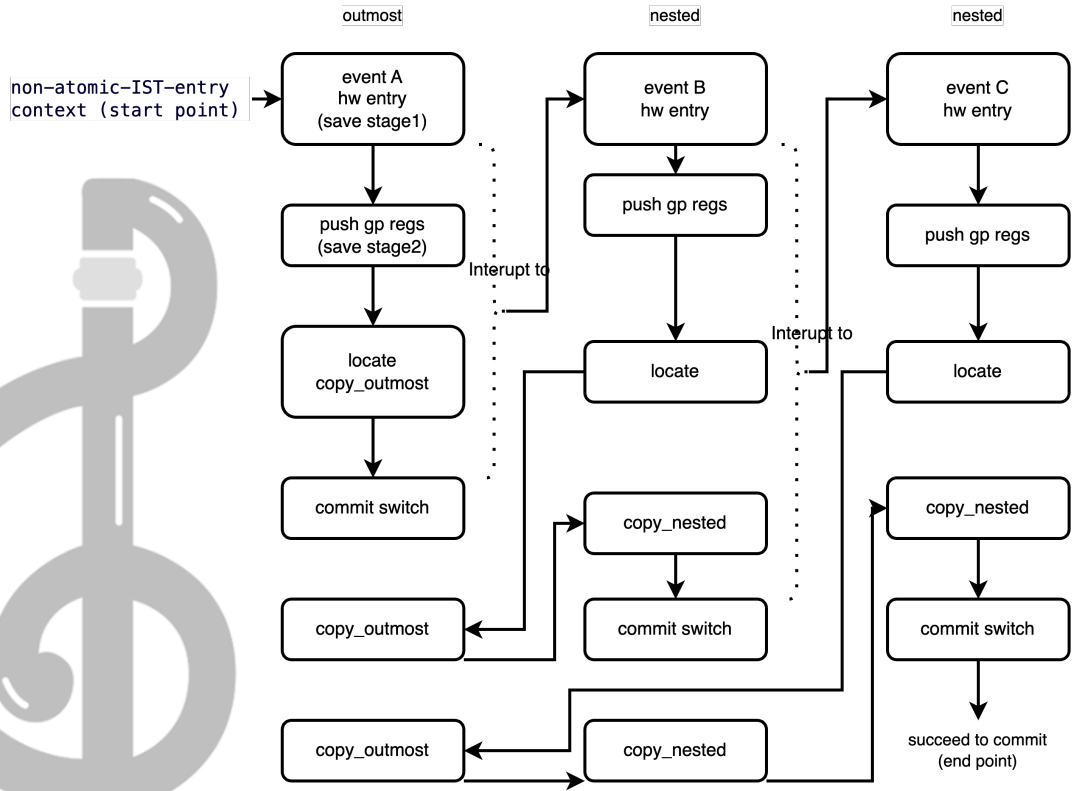
As an atomic entry, the innermost event's copy procedure should also act as a proxy to replicate the aborted copy procedures of the interrupted events, which amounts to copy all context for all events (interrupted and itself) to the main stack. What suppose to be copied varies depending on whether the event is the outmost or not. Define two sub procedures for it.

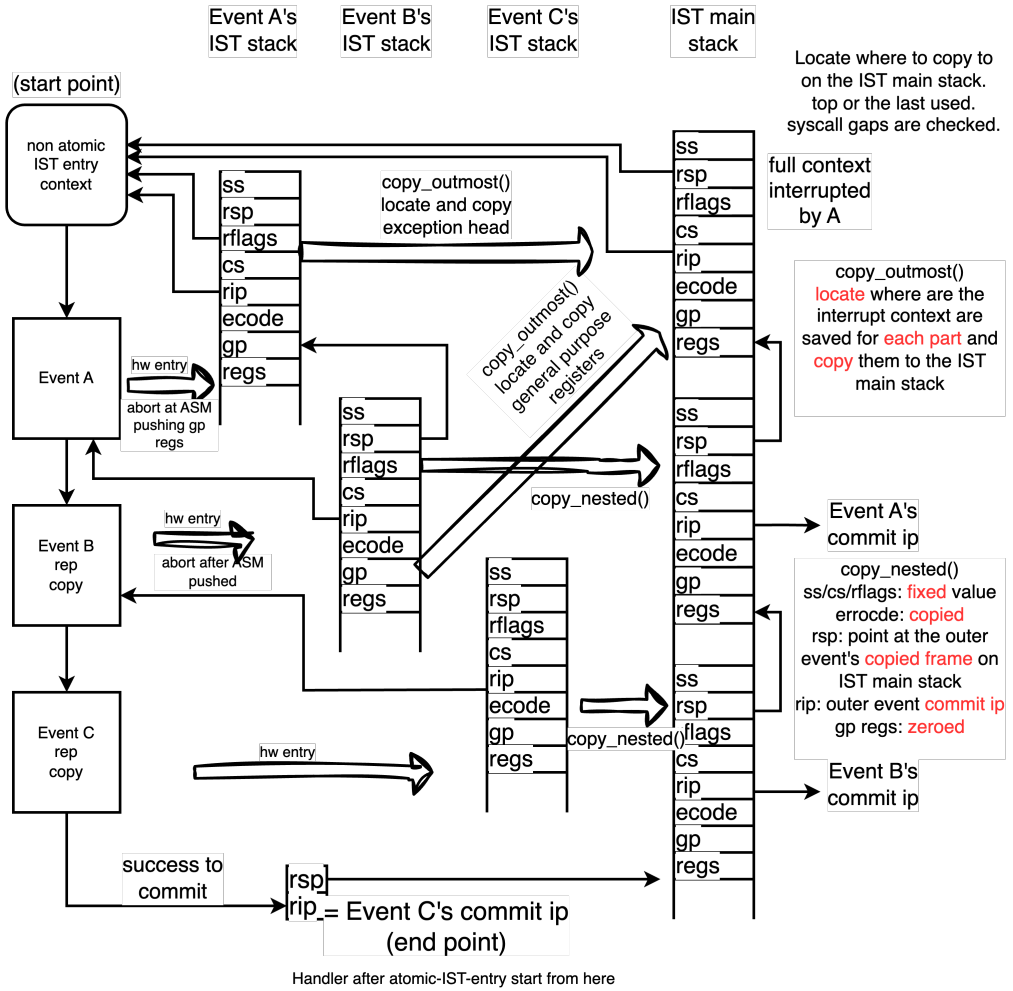
Save-Save-Copy(outmost)-Copy(nested)-Commit(switch)

Locate/identify: which is the outmost, where to copy to, where is the save stage1 saved, and where is the save stage2 saved.

Save-Save-Locate-Copy(outmost)-Copy(nested)-Commit(switch)

Atomic-IST-entry: full picture
Procedure view and Data view





Handler after atomic-IST-entry start from here

Replicative Bleeding

According to RetBleed mess (as PeterZ pointed out), you can NOT call a function and do a RET before:

```
IBRS_ENTER;
UNTRAIN_RET_FROM_CALL;
```

in entry code.

Three choices:

- No function call and no RET in atomic-IST-entry
- Make UNTRAIN_RET_FROM_CALL usable in early entry stage
- Add new stages to set kernel CR3/GSBASE/SPEC_CTRL in atomic-IST-entry

No function call and no RET

- Rewrite the C copy function with ASM
  - Not reviewable, Not maintainable
  - Contradict with the aim of conveting ASM code to C code

- Hack with return thunk
  - Create individual copy functions for each IST event to ensure each function is called from one place only
  - Compile them with -mfunction-return=thunk-extern or function_return("thunk-extern")
  - Patch those “jmp __x86_return_thunk” to jump to the respective only return address on build time or boot time.
  - Jump to those copy functions instead of calling to them from the entry ASM code

Make UNTRAIN_RET_FROM_CALL usable in early entry stage

- Split UNTRAIN_RET_FROM_CALL
  - A part is done before CR3/GSBASE/SPEC_CTR is switched
  - A part is after paranoid_entry()

- Pros
  - Keep the atomic-IST-entry simple

- Cons
  - The ASM version UNTRAIN_RET_FROM_CALL which is not in ist_entry.c has also to be in atomic-IST-entry, required to be Abortable, Idempotently-Replicable
  - Unsure if it can be possible

Add new stages to set kernel CR3/GSBASE/SPEC_CTRL

- Add save stage3 (ASM) to save system registers(CR3/GSBASE/SPEC_CTRL)
  - saving system registers clobbers gp registers, so it has to be a new save stage
- Locate stage also locates where save stage3 saved
- Copy(outmost) state also copy what save stage3 saved
- Add kernel-ize state to set CR3/GSBASE/SPEC_CTRL to kernel value which is also Abortable, Idempotently-Replicable.
- Copy(nested) copies the kernel values of CR3/GSBASE/SPEC_CTRL

Save-Save-Save-Locate-kernel-ize-Copy(outmost)-Copy(nested)-Commit(switch)

Pros:

- Implement paranoid_entry() in C which is also in our mission
- No more bleed

Cons:

- Too much code is in atomic-IST-entry, all of them is required to be Abortable, Idempotently-Replicable.
- Add C version UNTRAIN_RET_FROM_CALL.

Suggestion?

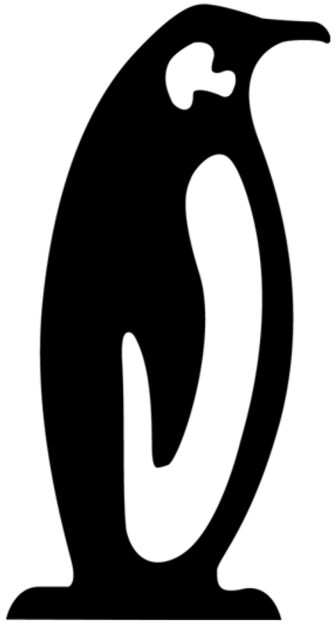
For I RETURN to update the code

Hack with return thunk or New stages?

Links

[tglx] https://lore.kernel.org/lkml/875z98jkof.fsf@nanos.tec.linutronix.de/
[linus] https://lore.kernel.org/lkml/CAHk-=wimnCtaDhCswqBUag37J1ALDno5dGv4v8Emv0b7SgVgPw@mail.gmail.com/
[andy] https://lore.kernel.org/lkml/CALCETrU9XypKbj-TrXLB3CPW6=MZ__5ifLz0ckbB=c=Myegn9Q@mail.gmail.com/
[andrew] https://docs.google.com/document/d/1hWejnyDkjRRAW-JEsRjA5c9CKLOPc6VKJQsuvODlQEI/edit#
[asm-to-c]: https://lore.kernel.org/lkml/20211126101209.8613-1-jiangshanlai@gmail.com/
[atomic-ist-entry] https://lore.kernel.org/lkml/20230403140605.540512-1-jiangshanlai@gmail.com/
[pvm] https://github.com/virt-pvm/linux/tree/pvm

Thanks!
Q&A



Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024