X86-64

Cauchy(pqy7172@gmail.com)

December 28, 2025

Contents

X86-64

```
=> e1000_intr
=> __handle_irq_event_percpu
=> handle_irq_event
=> handle_fasteoi_irq
=> __common_interrupt
=> common_interrupt
=> asm_common_interrupt
=> e1000_clean_rx_irq
=> e1000_clean
=> __napi_poll
=> net_rx_action
=> handle_softirqs
=> __irq_exit_rcu
=> common_interrupt
=> asm_common_interrupt
=> finish_task_switch.isra.0
=> __schedule
=> schedule
=> worker_thread
=> kthread
=> ret_from_fork
=> ret_from_fork_asm
```

1

X86-64 idt_setup_apic_and_irq_gates irq_entries_start X86-64

1.1

```
 .align IDT_ALIGN
SYM_CODE_START(irq_entries_start)
    vector=FIRST_EXTERNAL_VECTOR
    .rept NR_EXTERNAL_VECTORS
 UNWIND_HINT_IRET_REGS
0 :
 ENDBR
 .byte 0x6a, vector
 jmp asm_common_interrupt
 /* Ensure that the above is IDT_ALIGN bytes max */
 .fill 0b + IDT_ALIGN - ., 1, 0xcc
 vector = vector+1
    .endr
SYM_CODE_END(irq_entries_start)
```

NR_EXTERNAL_VECTORS FIRST_EXTERNAL_VECTOR

```
/*
 * Posted interrupt notification vector for all device MSIs delivered to
 * the host kernel.
 */
#define POSTED_MSI_NOTIFICATION_VECTOR 0xeb

/*
 * IDT vectors usable for external interrupt sources start at 0x20.
 * (0x80 is the syscall vector, 0x30-0x3f are for ISA)
 */
#define FIRST_EXTERNAL_VECTOR  0x20

#ifdef CONFIG_X86_LOCAL_APIC
#define FIRST_SYSTEM_VECTOR  POSTED_MSI_NOTIFICATION_VECTOR
#else
#define FIRST_SYSTEM_VECTOR  NR_VECTORS
#endif
#define NR_EXTERNAL_VECTORS  (FIRST_SYSTEM_VECTOR - FIRST_EXTERNAL_VECTOR)
```

X86-64 0x20 PIC APIC Local APIC CPU

x86-64 CONFIG_X86_LOCAL_APIC FIRST_SYSTEM_VECTOR POSTED_MSI_NOTIFICA
0xEB MSI Message Signaled Interrupts vCPU vCPU APICv APIC PI
Exit I/O POSTED_MSI_NOTIFICATION_VECTOR PI vCP
Exit vCPU VM-Exit

[0x20, 0xeb) 203 0x20 CPU 0 Page Fault NMI 0x20

.byte push push push 127 push+ 5 1 push 4 32 IDT
entry stub IDT_ALIGN

vmlinux irq_entries_start

```
ffffffff81e00230 <irq_entries_start>:
ffffffff81e00230:       f3 0f 1e fa             endbr64
ffffffff81e00234:       6a 20                   push   $0x20
ffffffff81e00236:       e9 05 13 00 00          jmp    ffffffff81e01540 <asm_common_interrup
ffffffff81e0023b:       cc                      int3
ffffffff81e0023c:       cc                      int3
ffffffff81e0023d:       cc                      int3
ffffffff81e0023e:       cc                      int3
ffffffff81e0023f:       cc                      int3
ffffffff81e00240:       f3 0f 1e fa             endbr64
ffffffff81e00244:       6a 21                   push   $0x21
ffffffff81e00246:       e9 f5 12 00 00          jmp    ffffffff81e01540 <asm_common_interrup
ffffffff81e0024b:       cc                      int3
ffffffff81e0024c:       cc                      int3
ffffffff81e0024d:       cc                      int3
ffffffff81e0024e:       cc                      int3
ffffffff81e0024f:       cc                      int3
```

push endbr64 X86_KERNEL_IBT
Control Protection Exception

.align IDT_ALIGN Intel IBT(Intel CET Control-flow Enforcement Technology) 16

.rept NR_EXTERNAL_VECTORS .

1.2

idt_setup_apic_and_irq_gates

```
/**
 * idt_setup_apic_and_irq_gates - Setup APIC/SMP and normal interrupt gates
 */
void __init idt_setup_apic_and_irq_gates(void)
{
 int i = FIRST_EXTERNAL_VECTOR;
 void *entry;

 idt_setup_from_table(idt_table, apic_idts, ARRAY_SIZE(apic_idts), true);
 for_each_clear_bit_from(i, system_vectors, FIRST_SYSTEM_VECTOR) {
  entry = irq_entries_start + IDT_ALIGN * (i - FIRST_EXTERNAL_VECTOR);
  set_intr_gate(i, entry);
 }

#ifdef CONFIG_X86_LOCAL_APIC
 for_each_clear_bit_from(i, system_vectors, NR_VECTORS) {
  /*
   * Don't set the non assigned system vectors in the
   * system_vectors bitmap. Otherwise they show up in
   * /proc/interrupts.
   */
  entry = spurious_entries_start + IDT_ALIGN * (i - FIRST_SYSTEM_VECTOR);
  set_intr_gate(i, entry);
 }
#endif
 /* Map IDT into CPU entry area and reload it. */
 idt_map_in_cea();
 load_idt(&idt_descr);

 /* Make the IDT table read only */
 set_memory_ro((unsigned long)&idt_table, 1);

 idt_setup_done = true;
}
```

apic_idts idt_table set_intr_gate idt_table idt_map_i
idt_setup_from_table

```
arch/x86/kernel/idt.c
static __init void
idt_setup_from_table(gate_desc *idt, const struct idt_data *t, int size, bool sys)
{
 gate_desc desc;

 for (; size > 0; t++, size--) {
  idt_init_desc(&desc, t);
  write_idt_entry(idt, t->vector, &desc);
  if (sys)
   set_bit(t->vector, system_vectors);
 }
}
```

/ apic_idts

```
arch/x86/kernel/idt.c
/*
```

```
 * The APIC and SMP idt entries
 */
static const __initconst struct idt_data apic_idts[] = {
#ifdef CONFIG_SMP
 INTG(RESCHEDULE_VECTOR,   asm_sysvec_reschedule_ipi),
 INTG(CALL_FUNCTION_VECTOR,  asm_sysvec_call_function),
 INTG(CALL_FUNCTION_SINGLE_VECTOR, asm_sysvec_call_function_single),
 INTG(REBOOT_VECTOR,   asm_sysvec_reboot),
#endif

#ifdef CONFIG_X86_THERMAL_VECTOR
 INTG(THERMAL_APIC_VECTOR,  asm_sysvec_thermal),
#endif

#ifdef CONFIG_X86_MCE_THRESHOLD
 INTG(THRESHOLD_APIC_VECTOR,  asm_sysvec_threshold),
#endif

#ifdef CONFIG_X86_MCE_AMD
 INTG(DEFERRED_ERROR_VECTOR,  asm_sysvec_deferred_error),
#endif

#ifdef CONFIG_X86_LOCAL_APIC
 INTG(LOCAL_TIMER_VECTOR,  asm_sysvec_apic_timer_interrupt),
 INTG(X86_PLATFORM_IPI_VECTOR,  asm_sysvec_x86_platform_ipi),
# if IS_ENABLED(CONFIG_KVM)
 INTG(POSTED_INTR_VECTOR,  asm_sysvec_kvm_posted_intr_ipi),
 INTG(POSTED_INTR_WAKEUP_VECTOR,  asm_sysvec_kvm_posted_intr_wakeup_ipi),
 INTG(POSTED_INTR_NESTED_VECTOR,  asm_sysvec_kvm_posted_intr_nested_ipi),
# endif
# ifdef CONFIG_IRQ_WORK
 INTG(IRQ_WORK_VECTOR,   asm_sysvec_irq_work),
# endif
 INTG(SPURIOUS_APIC_VECTOR,  asm_sysvec_spurious_apic_interrupt),
 INTG(ERROR_APIC_VECTOR,   asm_sysvec_error_interrupt),
# ifdef CONFIG_X86_POSTED_MSI
 INTG(POSTED_MSI_NOTIFICATION_VECTOR, asm_sysvec_posted_msi_notification),
# endif
#endif
};
```

asm_sysvec_reschedule_ipi

```
DECLARE_IDTENTRY(RESCHEDULE_VECTOR,   sysvec_reschedule_ipi);
```

DECLARE_IDTENTRY .c .S .c

```
arch/x86/include/asm/idtentry.h
```

```
/**
 * DECLARE_IDTENTRY - Declare functions for simple IDT entry points
 *        No error code pushed by hardware
 * @vector: Vector number (ignored for C)
 * @func: Function name of the entry point
 *
 * Declares four functions:
 * - The ASM entry point: asm_##func
 * - The XEN PV trap entry point: xen_##func (maybe unused)
 * - The C handler called from the FRED event dispatcher (maybe unused)
 * - The C handler called from the ASM entry point
 *
 * Note: This is the C variant of DECLARE_IDTENTRY(). As the name says it
 * declares the entry points for usage in C code. There is an ASM variant
 * as well which is used to emit the entry stubs in entry_32/64.S.
 */
#define DECLARE_IDTENTRY(vector, func)     \
 asmlinkage void asm_##func(void);    \
 asmlinkage void xen_asm_##func(void);    \
 void fred_##func(struct pt_regs *regs);    \
 __visible void func(struct pt_regs *regs)
```

asm_sysvec_reschedule_ipi .S asm_sysvec_reschedule_ipi

```
arch/x86/include/asm/idtentry.h

#else /* !__ASSEMBLER__ */

/*
 * The ASM variants for DECLARE_IDTENTRY*() which emit the ASM entry stubs.
 */
#define DECLARE_IDTENTRY(vector, func)     \
 idtentry vector asm_##func func has_error_code=0
```

identry arch/x86/entry/entry_64.S asm_##func asm_sysvec_reschedule_ipi idtentry ent

```
ffffffff81e01630 <asm_sysvec_reschedule_ipi>:
ffffffff81e01630:       f3 0f 1e fa             endbr64
ffffffff81e01634:       90                      nop
ffffffff81e01635:       90                      nop
ffffffff81e01636:       90                      nop
ffffffff81e01637:       fc                      cld
ffffffff81e01638:       6a ff                   push   $0xffffffffffffffff
ffffffff81e0163a:       e8 f1 05 00 00          call   ffffffff81e01c30 <error_entry>
ffffffff81e0163f:       48 89 c4                mov    %rax,%rsp
ffffffff81e01642:       48 89 e7                mov    %rsp,%rdi
ffffffff81e01645:       e8 36 35 ef ff          call   ffffffff81cf4b80 <sysvec_reschedule_i
```

```
ffffffff81e0164a:       e9 21 07 00 00          jmp    ffffffff81e01d70 <error_return>
ffffffff81e0164f:       90                      nop
```

asm_sysvec_reboot

```
ffffffff81e01650 <asm_sysvec_reboot>:
ffffffff81e01650:       f3 0f 1e fa             endbr64
ffffffff81e01654:       90                      nop
ffffffff81e01655:       90                      nop
ffffffff81e01656:       90                      nop
ffffffff81e01657:       fc                      cld
ffffffff81e01658:       6a ff                   push   $0xffffffffffffffff
ffffffff81e0165a:       e8 d1 05 00 00          call   ffffffff81e01c30 <error_entry>
ffffffff81e0165f:       48 89 c4                mov    %rax,%rsp
ffffffff81e01662:       48 89 e7                mov    %rsp,%rdi
ffffffff81e01665:       e8 86 34 ef ff          call   ffffffff81cf4af0 <sysvec_reboot>
ffffffff81e0166a:       e9 01 07 00 00          jmp    ffffffff81e01d70 <error_return>
ffffffff81e0166f:       90                      nop
```

error_entry sysvec_reschedule_ipi/sysvec_reboot error_return call sysvec_reschedule_ipi

```
identry->idtentry_body->call \cfunc
```

cfunc identry cfunc DECLARE_IDTENTRY sysvec_reschedule_ipi

```
DECLARE_IDTENTRY(RESCHEDULE_VECTOR,   sysvec_reschedule_ipi);
```

call sysvec_reschedule_ipi

```
arch/x86/kernel/smp.c

DEFINE_IDTENTRY_SYSVEC_SIMPLE(sysvec_reschedule_ipi)
{
 apic_eoi();
 trace_reschedule_entry(RESCHEDULE_VECTOR);
 inc_irq_stat(irq_resched_count);
 scheduler_ipi();
 trace_reschedule_exit(RESCHEDULE_VECTOR);
}
```

```
arch/x86/include/asm/idtentry.h

#define DEFINE_IDTENTRY_SYSVEC_SIMPLE(func)    \
static __always_inline void __##func(struct pt_regs *regs);  \
         \
static __always_inline void instr_##func(struct pt_regs *regs)  \
{         \
 __irq_enter_raw();      \
 __##func (regs);      \
 __irq_exit_raw();      \
}         \
         \
__visible noinstr void func(struct pt_regs *regs)   \
{         \
 irqentry_state_t state = irqentry_enter(regs);   \
         \
 kvm_set_cpu_l1tf_flush_l1d();                                  \
 instrumentation_begin();     \
 instr_##func (regs);      \
 instrumentation_end();      \
 irqentry_exit(regs, state);     \
}         \
         \
void fred_##func(struct pt_regs *regs)     \
{         \
 instr_##func (regs);      \
}         \
         \
static __always_inline void __##func(struct pt_regs *regs)
```

___##func

```
{
 apic_eoi();
 trace_reschedule_entry(RESCHEDULE_VECTOR);
 inc_irq_stat(irq_resched_count);
 scheduler_ipi();
 trace_reschedule_exit(RESCHEDULE_VECTOR);
}
```

identry / cfunc sysvec_reschedule_ipi

```
DECLARE_IDTENTRY_SYSVEC(REBOOT_VECTOR,   sysvec_reboot);
```

/ XXX funcxxx

```
DECLARE_IDTENTRY_SYSVEC(XXX, funcxxx)
```

DEFINE_IDTENTRY_SYSVEC_SIMPLE

```
DEFINE_IDTENTRY_SYSVEC_SIMPLE(funcxxx)
```

sysvec_reschedule_ipi scheduler_ipi

```
DECLARE_IDTENTRY_SYSVEC->DECLARE_IDTENTRY->idtentry
```

asm_sysvec_reschedule_ipi/asm_sysvec_reboot addr INTG

```
#define G(_vector, _addr, _ist, _type, _dpl, _segment) \
 {      \
  .vector  = _vector,  \
  .bits.ist = _ist,   \
  .bits.type = _type,  \
  .bits.dpl = _dpl,   \
  .bits.p  = 1,   \
  .addr  = _addr,  \
  .segment = _segment,  \
 }

/* Interrupt gate */
#define INTG(_vector, _addr)    \
 G(_vector, _addr, DEFAULT_STACK, GATE_INTERRUPT, DPL0, __KERNEL_CS)
```

idt_data::addr apic_idts apic_idts 0-31 cpu 32-NR_EXTERNAL_VECTORS idt_setup_apic_and_irq_gates->idt_setup_from_table

```
static __init void
idt_setup_from_table(gate_desc *idt, const struct idt_data *t, int size, bool sys)
{
 gate_desc desc;

 for (; size > 0; t++, size--) {
  idt_init_desc(&desc, t);
  write_idt_entry(idt, t->vector, &desc);
  if (sys)
   set_bit(t->vector, system_vectors);
 }
}
```

apic_idts idt_init_desc

```
static inline void idt_init_desc(gate_desc *gate, const struct idt_data *d)
{
 unsigned long addr = (unsigned long) d->addr;

 gate->offset_low = (u16) addr;
 gate->segment  = (u16) d->segment;
 gate->bits  = d->bits;
 gate->offset_middle = (u16) (addr >> 16);
#ifdef CONFIG_X86_64
 gate->offset_high = (u32) (addr >> 32);
 gate->reserved  = 0;
#endif
}
```

low/middle/high gate_desc write_idt_entry idt_table native_write_id

```
static inline void native_write_idt_entry(gate_desc *idt, int entry, const gate_desc *gate)
{
 memcpy(&idt[entry], gate, sizeof(*gate));
}
```

entry idt_table system_vectors bitmap sys
vector true apic_idts

```
for_each_clear_bit_from(i, system_vectors, FIRST_SYSTEM_VECTOR) {
 entry = irq_entries_start + IDT_ALIGN * (i - FIRST_EXTERNAL_VECTOR);
 set_intr_gate(i, entry);
}
```

FIRST_SYSTEM_VECTOR system_vectors irq_entries_start set_intr_gate id

```
static __init void set_intr_gate(unsigned int n, const void *addr)
{
 struct idt_data data;

 init_idt_data(&data, n, addr);

 idt_setup_from_table(idt_table, &data, 1, false);
}
```

init_idt_data idt_data idt_setup_from_table irq_entries_start idt_table false
i system_vectors idt_table / comm
>spurious_interrupt CPU “ ” handle_spurious_interrup
interrupt /proc/interrupts APIC End-Of-Interrupt(EOI)
spurious_entries_start

```
./arch/x86/include/asm/idtentry.h
SYM_CODE_START(spurious_entries_start)
    vector=FIRST_SYSTEM_VECTOR
    .rept NR_SYSTEM_VECTORS
 UNWIND_HINT_IRET_REGS
0 :
 ENDBR
 .byte 0x6a, vector
 jmp asm_spurious_interrupt
 /* Ensure that the above is IDT_ALIGN bytes max */
 .fill 0b + IDT_ALIGN - ., 1, 0xcc
 vector = vector+1
    .endr
SYM_CODE_END(spurious_entries_start)
```

asm_spurious_interrupt asm_sysvec_reschedule_ipi C

```
arch/x86/include/asm/idtentry.h

DECLARE_IDTENTRY_IRQ(X86_TRAP_OTHER, spurious_interrupt);

#define DECLARE_IDTENTRY_IRQ(vector, func)    \
 DECLARE_IDTENTRY_ERRORCODE(vector, func)

#define DECLARE_IDTENTRY_ERRORCODE(vector, func)   \
 asmlinkage void asm_##func(void);    \
 asmlinkage void xen_asm_##func(void);    \
 __visible void func(struct pt_regs *regs, unsigned long error_code)
```

.S

```
arch/x86/include/asm/idtentry.h

#define DECLARE_IDTENTRY_ERRORCODE(vector, func)   \
 idtentry vector asm_##func func has_error_code=1
```

ipi
has_error_code 1 orig_ax
idt_map_in_cea

```
static void __init idt_map_in_cea(void)
{
 /*
  * Set the IDT descriptor to a fixed read-only location in the cpu
  * entry area, so that the "sidt" instruction will not leak the
  * location of the kernel, and to defend the IDT against arbitrary
  * memory write vulnerabilities.
  */
 cea_set_pte(CPU_ENTRY_AREA_RO_IDT_VADDR, __pa_symbol(idt_table),
      PAGE_KERNEL_RO);
 idt_descr.address = CPU_ENTRY_AREA_RO_IDT;
}
```

CPU_ENTRY_AREA_RO_IDT_VADDR

```
arch/x86/include/asm/pgtable_areas.h
#define CPU_ENTRY_AREA_RO_IDT_VADDR ((void *)CPU_ENTRY_AREA_RO_IDT)

/* Single page reserved for the readonly IDT mapping: */
#define CPU_ENTRY_AREA_RO_IDT  CPU_ENTRY_AREA_BASE

arch/x86/include/asm/pgtable_64_types.h
#define CPU_ENTRY_AREA_BASE (CPU_ENTRY_AREA_PGD << P4D_SHIFT)

#define CPU_ENTRY_AREA_PGD _AC(-4, UL)
#define P4D_SHIFT  39
```

CPU_ENTRY_AREA_RO_IDT_VADDR 0xfffffe0000000000 x86-64 Documentation/arch/x86/x86_64/mm.rst 0000000000000000-00007fffffffefff 128TB ffffc90
ffffe8ffffffffff 32TB vmalloc/ioremap

CPU_ENTRY_AREA_RO_IDT_VADDR fffffe0000000000 fffffe0000000000-fffffe7fffffffff 512GB cpu_entry_area mapping idt Page-Global cr3 page global tlb cea_set_pte

```
void cea_set_pte(void *cea_vaddr, phys_addr_t pa, pgprot_t flags)
{
 unsigned long va = (unsigned long) cea_vaddr;
 pte_t pte = pfn_pte(pa >> PAGE_SHIFT, flags);

 /*
  * The cpu_entry_area is shared between the user and kernel
  * page tables.  All of its ptes can safely be global.
  * _PAGE_GLOBAL gets reused to help indicate PROT_NONE for
  * non-present PTEs, so be careful not to set it in that
  * case to avoid confusion.
  */
 if (boot_cpu_has(X86_FEATURE_PGE) &&
     (pgprot_val(flags) & _PAGE_PRESENT))
  pte = pte_set_flags(pte, _PAGE_GLOBAL);

 set_pte_vaddr(va, pte);
}
```

idt_table idt_table idt_table __pa_symbol

```
arch/x86/include/asm/page.h

#define __pa_symbol(x) \
 __phys_addr_symbol(__phys_reloc_hide((unsigned long)(x)))
```

```
arch/x86/include/asm/page_64.h

#define __phys_addr_symbol(x) \
 ((unsigned long)(x) - __START_KERNEL_map + phys_base)
```

__START_KERNEL_map phys_base x86-64 __START_KERNEL_map

```
x86/include/asm/page_64_types.h

#define __START_KERNEL_map _AC(0xffffffff80000000, UL)
```

Documentation/arch/x86/x86_64/mm.rst

```
ffffffff80000000 |   -2    GB | ffffffff9fffffff |  512 MB | kernel text mapping, mapped to
```

idt_table __phys_addr_symbol

```
/* Must be page-aligned because the real IDT is used in the cpu entry area */
static gate_desc idt_table[IDT_ENTRIES] __page_aligned_bss;
```

phys_base __START_KERNEL_map phys_base phys_base __startup_64

```
/*
 * Compute the delta between the address I am compiled to run at
 * and the address I am actually running at.
 */
phys_base = load_delta = __START_KERNEL_map + p2v_offset;
```

__startup_64 startup_64

arch/x86/kernel/head_64.S

```
 call __startup_64
```

startup_64 __startup_64 p2v_offset phys_base KASLR

```
p2v_offset = phys_base - __START_KERNEL_map
```

startup_64 rip p2v_offset __startup_64 C

```
phys_base = __START_KERNEL_map + p2v_offset
```

__START_KERNEL_map phys_base virt addr __START_KERNEL_map phys_base virt addr/phys addr __startup_64 [__ST
MMU CR3 phys_base cpu Page Fault
cea_set_pte pte PAGE_KERNEL_RO

arch/x86/include/asm/pgtable_types.h

```
#define PAGE_KERNEL_RO  __pgprot_mask(__PAGE_KERNEL_RO        | _ENC)
```

_ENC AMD __pgprot_mask

arch/x86/include/asm/pgtable_types.h

```
#define __pgprot_mask(x) __pgprot((x) & __default_kernel_pte_mask)
#define __pg(x)   __pgprot(x)
#define __pgprot(x)  ((pgprot_t) { (x) } )
typedef struct pgprot { pgprotval_t pgprot; } pgprot_t;
```

arch/x86/include/asm/pgtable_64_types.h

```
typedef unsigned long pgprotval_t;
```

__pgprot_mask unsigned long bit __default_kernel_pte_mask _PAGE_GLOBAL

```
arch/x86/mm/init.c: probe_page_size_mask

 /* Except when with PTI where the kernel is mostly non-Global: */
 if (cpu_feature_enabled(X86_FEATURE_PTI))
  __default_kernel_pte_mask &= ~_PAGE_GLOBAL;
```

```
arch/x86/mm/init_64.c

/* Bits allowed in normal kernel mappings: */
pteval_t __default_kernel_pte_mask __read_mostly = ~0;p
```

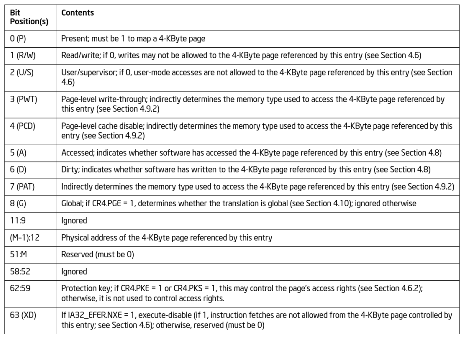
___PAGE_KERNEL_RO

```
arch/x86/include/asm/pgtable_types.h

#define __PAGE_KERNEL_RO  (__PP|   0|   0|___A|__NX|   0|   0|___G)
```

___PP ___PP

```
arch/x86/include/asm/pgtable_types.h

#define __PP _PAGE_PRESENT
#define _PAGE_PRESENT (_AT(pteval_t, 1) << _PAGE_BIT_PRESENT)
#define _PAGE_BIT_PRESENT 0 /* is present */
```

bit x86-64 4KB intel sdm vol3 bit



Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 4.6.2); otherwise, it is not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Figure 1: 4KB

cea_set_pte cea_set_pte

```
void cea_set_pte(void *cea_vaddr, phys_addr_t pa, pgprot_t flags)
{
 unsigned long va = (unsigned long) cea_vaddr;
 pte_t pte = pfn_pte(pa >> PAGE_SHIFT, flags);

 /*
```

```
  * The cpu_entry_area is shared between the user and kernel
  * page tables.  All of its ptes can safely be global.
  * _PAGE_GLOBAL gets reused to help indicate PROT_NONE for
  * non-present PTEs, so be careful not to set it in that
  * case to avoid confusion.
  */
 if (boot_cpu_has(X86_FEATURE_PGE) &&
     (pgprot_val(flags) & _PAGE_PRESENT))
  pte = pte_set_flags(pte, _PAGE_GLOBAL);

 set_pte_vaddr(va, pte);
}
```

pte_t :

```
typedef struct { pteval_t pte; } pte_t;
```

pteval_t 32/64 bit 64 unsigned long

```
typedef unsigned long pteval_t;
```

pte entry 64 bit pfn_pte mm pte

```
static inline pte_t pfn_pte(unsigned long page_nr, pgprot_t pgprot)
{
 phys_addr_t pfn = (phys_addr_t)page_nr << PAGE_SHIFT;
 /* This bit combination is used to mark shadow stacks */
 WARN_ON_ONCE((pgprot_val(pgprot) & (_PAGE_DIRTY | _PAGE_RW)) ==
   _PAGE_DIRTY);
 pfn ^= protnone_mask(pgprot_val(pgprot));
 pfn &= PTE_PFN_MASK;
 return __pte(pfn | check_pgprot(pgprot));
}
```

page_nr PAGE_SHIFT pte 12 page_nr idt_table PAGE_SH
pfn_pte pgprot _PAGE_DIRTY _PAGE_RW “ ” shadow stacks CET feature (_PAGE_RW) CPU _PAGE_DIRTY (!_PAGE_RW) _PAGE_
_PAGE_DIRTY

```
/*
 * A clear pte value is special, and doesn't get inverted.
 *
 * Note that even users that only pass a pgprot_t (rather
 * than a full pte) won't trigger the special zero case,
 * because even PAGE_NONE has _PAGE_PROTNONE | _PAGE_ACCESSED
 * set. So the all zero case really is limited to just the
 * cleared page table entry case.
 */
```

```
static inline bool __pte_needs_invert(u64 val)
{
 return val && !(val & _PAGE_PRESENT);
}

/* Get a mask to xor with the page table entry to get the correct pfn. */
static inline u64 protnone_mask(u64 val)
{
 return __pte_needs_invert(val) ?  ~0ull : 0;
}
```

x86 64 PTE (PFN) PFN + mprotect(PROT_NONE) “ ” Linux
PROTNONE pfn invert PFN “ ” “PROTNONE” val
!= 0 val & _PAGE_PRESENT!= 0 ___pte_needs_invert(val) false pte PFN invert val
= = 0 “ ” pte_clear invert val != 0 val & _PAGE_PRESENT
== 0 mprotect(PROT_NONE) “ ” PFN ___pte_needs_invert(val) true invert PF

```
pfn ^= protnone_mask(pgprot_val(pgprot));
```

PROTNONE PFN ___PP protnone_mask 0 0

```
/* Extracts the PFN from a (pte|pmd|pud|pgd)val_t of a 4KB page */
#define PTE_PFN_MASK  ((pteval_t)PHYSICAL_PAGE_MASK)
```

4KB PFN 64 PHYSICAL_PAGE_MASK

```
#define PHYSICAL_PAGE_MASK (((signed long)PAGE_MASK) & __PHYSICAL_MASK)
```

PAGE_MASK

```
#define PAGE_MASK (~(PAGE_SIZE - 1))
```

___PHYSICAL_MASK bit

```
#define __PHYSICAL_MASK  physical_mask
```

```
phys_addr_t physical_mask __ro_after_init = (1ULL << __PHYSICAL_MASK_SHIFT) - 1;
#define __PHYSICAL_MASK_SHIFT 52
```

___pte

```
#define __pte(x) native_make_pte(x)
```

```
static inline pte_t native_make_pte(pteval_t val)
{
 return (pte_t) { .pte = val };
}
```

pfn_pte cea_set_pte present=1 global present=0 global Linux “ ”
present global PROT_NONE present global _PAGE_GLOBAL CR3 flu
pte_set_flags

```
static inline pte_t pte_set_flags(pte_t pte, pteval_t set)
{
 pteval_t v = native_pte_val(pte);

 return native_make_pte(v | set);
}

static inline pte_t native_make_pte(pteval_t val)
{
 return (pte_t) { .pte = val };
}
```

va pte set_pte_vaddr pte

```
arch/x86/mm/init_64.c

void set_pte_vaddr(unsigned long vaddr, pte_t pteval)
{
 pgd_t *pgd;
 p4d_t *p4d_page;

 pr_debug("set_pte_vaddr %lx to %lx\n", vaddr, native_pte_val(pteval));

 pgd = pgd_offset_k(vaddr);
 if (pgd_none(*pgd)) {
  printk(KERN_ERR
   "PGD FIXMAP MISSING, it should be setup in head.S!\n");
  return;
 }

 p4d_page = p4d_offset(pgd, 0);
 set_pte_vaddr_p4d(p4d_page, vaddr, pteval);
}
```

set_pte_vaddr crash vtop vtop 0xfffffe000000

```
crash> vtop 0xfffffe0000000000
VIRTUAL           PHYSICAL
fffffe0000000000  5db2d000

PGD DIRECTORY: ffffffffb3422000
PAGE DIRECTORY: 23ffc5067
   PUD: 23ffc5000 => 1003b8067
   PMD: 1003b8000 => 1003b9067
   PTE: 1003b9000 => 800000005db2d121
  PAGE: 5db2d000
```

```
      PTE         PHYSICAL  FLAGS
800000005db2d121  5db2d000  (PRESENT|ACCESSED|GLOBAL|NX)

      PAGE        PHYSICAL      MAPPING       INDEX CNT FLAGS
ffffde8e4176cb40  5db2d000                0        0  1 fffffc0002000 reserved
```

pgd_offset_k   init_mm   address pgd

```
include/linux/pgtable.h

/*
 * a shortcut which implies the use of the kernel's pgd, instead
 * of a process's
 */
#define pgd_offset_k(address)  pgd_offset(&init_mm, (address))
#define pgd_offset(mm, address)  pgd_offset_pgd((mm)->pgd, (address))
static inline pgd_t *pgd_offset_pgd(pgd_t *pgd, unsigned long address)
{
 return (pgd + pgd_index(address));
};
#define pgd_index(a)  (((a) >> PGDIR_SHIFT) & (PTRS_PER_PGD - 1))
```

```
arch/x86/include/asm/pgtable_64_types.h

/*
 * PGDIR_SHIFT determines what a top-level page table entry can map
 */
#define PGDIR_SHIFT pgdir_shift
#define PTRS_PER_PGD 512
```

pgdir_shift   39

```
arch/x86/kernel/head64.c

unsigned int pgdir_shift __ro_after_init = 39;
```

5 pgdir_shift   crash p pgdir_shift   39

```
crash> p pgdir_shift
pgdir_shift = $29 = 39
```

pgd_offset_k   pgd_offset_k

```
init_mm.pgd + (address >> 39 & 511) * 8
```

init_mm.pgd

```
mm/init-mm.c
```

```
struct mm_struct init_mm = {
    ...
    .pgd  = swapper_pg_dir,
    ...
};
```

```
arch/x86/include/asm/pgtable_64.h

#define swapper_pg_dir init_top_pgt
```

init_top_pgt PGD

```
arch/x86/kernel/head_64.S

SYM_DATA_START_PTI_ALIGNED(init_top_pgt)
 .quad   level3_ident_pgt - __START_KERNEL_map + _KERNPG_TABLE_NOENC
 .org    init_top_pgt + L4_PAGE_OFFSET*8, 0
 .quad   level3_ident_pgt - __START_KERNEL_map + _KERNPG_TABLE_NOENC
 .org    init_top_pgt + L4_START_KERNEL*8, 0
 /* (2^48-(2*1024*1024*1024))/(2^39) = 511 */
 .quad   level3_kernel_pgt - __START_KERNEL_map + _PAGE_TABLE_NOENC
 .fill PTI_USER_PGD_FILL,8,0
SYM_DATA_END(init_top_pgt)
```

.quad cpu .quad level3_ident_pg/level3_kernel_pgt crash pgd init_top_pgt

```
crash> p init_mm.pgd
$30 = (pgd_t *) 0xffffffffb3422000 <init_top_pgt>
```

8 1 8

```
0xffffffffb3422000 + (0xfffffe0000000000 >> 39 & 511) * 8 = 0xffffffffb3422fe0
```

crash rd crash 23ffc5067

```
crash> rd 0xffffffffb3422fe0
ffffffffb3422fe0:  000000023ffc5067                    gP.?....
```

pgd_offset_k pgd head_64.S p4d_offset

```
arch/x86/include/asm/pgtable.h

/* to find an entry in a page-table-directory. */
static inline p4d_t *p4d_offset(pgd_t *pgd, unsigned long address)
{
 if (!pgtable_l5_enabled())
  return (p4d_t *)pgd;
 return (p4d_t *)pgd_page_vaddr(*pgd) + p4d_index(address);
}
```

AMD 5 cat /proc/self/maps 12 0 4bit 48bit

arch/x86/include/asm/pgtable.h

```
static inline unsigned long pgd_page_vaddr(pgd_t pgd)
{
 return (unsigned long)__va((unsigned long)pgd_val(pgd) & PTE_PFN_MASK);
}
```

arch/x86/include/asm/page.h

```
#define __va(x)   ((void *)((unsigned long)(x)+PAGE_OFFSET))
```

arch/x86/include/asm/page_types.h

```
#define PAGE_OFFSET  ((unsigned long)__PAGE_OFFSET)
```

arch/x86/include/asm/page_64_types.h

```
#define __PAGE_OFFSET           page_offset_base
```

page_offset_base arch/x86/mm/kaslr.c PAGE_OFFSET crash page_offset_base

```
crash> p/x page_offset_base
$38 = 0xffff8a92c0000000
```

*pgd pgd p4d PTE_PFN_MASK PTE_PFN_MASK

```
p4d_page = 0xffff8a92c0000000+0x23ffc5000 = 0xffff8a94fffc5000
```

crash rd p4d pud

```
crash> rd 0xffff8a94fffc5000
ffff8a94fffc5000:  00000001003b8067                    g.;.....
```

crash

```
PUD: 23ffc5000 => 1003b8067
```

set_pte_vaddr_p4d

arch/x86/mm/init_64.c

```
void set_pte_vaddr_p4d(p4d_t *p4d_page, unsigned long vaddr, pte_t new_pte)
{
 p4d_t *p4d = p4d_page + p4d_index(vaddr);
 pud_t *pud = fill_pud(p4d, vaddr);

 __set_pte_vaddr(pud, vaddr, new_pte);
}
```

p4d_index

```
arch/x86/include/asm/pgtable.h

static inline unsigned long p4d_index(unsigned long address)
{
 return (address >> P4D_SHIFT) & (PTRS_PER_P4D - 1);
}

arch/x86/include/asm/pgtable_64_types.h

#define PTRS_PER_P4D  ptrs_per_p4d

arch/x86/boot/compressed/pgtable_64.c

unsigned int __section(".data") ptrs_per_p4d = 1;
```

crash ptrs_per_p4d 1 p4d_index 0 p4d p4d p4d_page 0xffff8a94fffc5000 fill_pud

```
arch/x86/mm/init_64.c

static pud_t *fill_pud(p4d_t *p4d, unsigned long vaddr)
{
 if (p4d_none(*p4d)) {
  pud_t *pud = (pud_t *)spp_getpage();
  p4d_populate(&init_mm, p4d, pud);
  if (pud != pud_offset(p4d, 0))
   printk(KERN_ERR "PAGETABLE BUG #01! %p <-> %p\n",
          pud, pud_offset(p4d, 0));
 }
 return pud_offset(p4d, vaddr);
}
```

p4d p4d flag spp_getpage pud

```
/*
 * NOTE: This function is marked __ref because it calls __init function
 * (alloc_bootmem_pages). It's safe to do it ONLY when after_bootmem == 0.
 */
static __ref void *spp_getpage(void)
{
 void *ptr;

 if (after_bootmem)
  ptr = (void *) get_zeroed_page(GFP_ATOMIC);
 else
  ptr = memblock_alloc(PAGE_SIZE, PAGE_SIZE);
 if (!ptr || ((unsigned long)ptr & ~PAGE_MASK)) {
  panic("set_pte_phys: cannot allocate page data %s\n",
   after_bootmem ? "after bootmem" : "");
 }

 pr_debug("spp_getpage %p\n", ptr);

 return ptr;
}
```

__ref init/exit / init/exit init/exit modpost

```
/*
 * modpost check for section mismatches during the kernel build.
 * A section mismatch happens when there are references from a
 * code or data section to an init section (both code or data).
 * The init sections are (for most archs) discarded by the kernel
 * when early init has completed so all such references are potential bugs.
 * For exit sections the same issue exists.
 *
 * The following markers are used for the cases where the reference to
 * the *init / *exit section (code or data) is valid and will teach
 * modpost not to issue a warning.  Intended semantics is that a code or
 * data tagged __ref* can reference code or data from init section without
 * producing a warning (of course, no warning does not mean code is
 * correct, so optimally document why the __ref is needed and why it's OK).
 *
 * The markers follow same syntax rules as __init / __initdata.
 */
#define __ref            __section(".ref.text") noinline
```

spp_getpage

```
/*
 * NOTE: This function is marked __ref because it calls __init function
 * (alloc_bootmem_pages). It's safe to do it ONLY when after_bootmem == 0.
 */
static __ref void *spp_getpage(void)
{
 void *ptr;

 if (after_bootmem)
  ptr = (void *) get_zeroed_page(GFP_ATOMIC);
 else
  ptr = memblock_alloc(PAGE_SIZE, PAGE_SIZE);
 if (!ptr || ((unsigned long)ptr & ~PAGE_MASK)) {
  panic("set_pte_phys: cannot allocate page data %s\n",
   after_bootmem ? "after bootmem" : "");
 }

 pr_debug("spp_getpage %p\n", ptr);

 return ptr;
}
```

memblock memblock_alloc buddy system get_zeroed_page p4d_populate p4d

```
arch/x86/include/asm/pgalloc.h

static inline void p4d_populate(struct mm_struct *mm, p4d_t *p4d, pud_t *pud)
{
 paravirt_alloc_pud(mm, __pa(pud) >> PAGE_SHIFT);
 set_p4d(p4d, __p4d(_PAGE_TABLE | __pa(pud)));
}
```

_PAGE_TABLE pte

```
/*
 * Page tables needs to have Write=1 in order for any lower PTEs to be
 * writable. This includes shadow stack memory (Write=0, Dirty=1)
 */

#define _PAGE_TABLE   (__PP|__RW|_USR|___A|   0|___D|   0|   0| _ENC)
```

page table ___pa

```
arch/x86/include/asm/page.h

#ifndef __pa
#define __pa(x)  __phys_addr((unsigned long)(x))
#endif
```

```
arch/x86/include/asm/page_64.h

#define __phys_addr(x)  __phys_addr_nodebug(x)

static __always_inline unsigned long __phys_addr_nodebug(unsigned long x)
{
 unsigned long y = x - __START_KERNEL_map;

 /* use the carry flag to determine if x was < __START_KERNEL_map */
 x = y + ((x > y) ? phys_base : (__START_KERNEL_map - PAGE_OFFSET));

 return x;
}
```

___phys_addr_nodebug ___phys_addr_symbol x > ___START_KERNEL_map = x - ___START_KERNEL_map + phy_base x < ___START_KERNEL_map y x > y phy_addr(x) = x - ___START_KERNEL_map + ___START_KERNEL_map - PAGE_OFFSET = x - PAGE_OFFSET PAGE_OFFSET PAGE_OFFSET ___p4d flag _PAGE_TABLE

```
arch/x86/include/asm/pgtable.h

#ifndef set_p4d
# define set_p4d(p4dp, p4d)  native_set_p4d(p4dp, p4d)
#endif

static inline void native_set_p4d(p4d_t *p4dp, p4d_t p4d)
{
 pgd_t pgd;

 if (pgtable_l5_enabled() ||
     !IS_ENABLED(CONFIG_MITIGATION_PAGE_TABLE_ISOLATION)) {
  WRITE_ONCE(*p4dp, p4d);
  return;
 }

 pgd = native_make_pgd(native_p4d_val(p4d));
 pgd = pti_set_user_pgtbl((pgd_t *)p4dp, pgd);
 WRITE_ONCE(*p4dp, native_make_p4d(native_pgd_val(pgd)));
}
```

WRITE_ONCE pti / p4d p4dp WRITE_ONCE *p4dp p4dp ___set_pte_vaddr spp_getpage CPU_ENTRY_AREA_RO_IDT_V cea_set_pte / crash vtop 0xfffffe0000000000

```
crash> vtop 0xfffffe0000000000
VIRTUAL           PHYSICAL
fffffe0000000000  5db2d000

PGD DIRECTORY: ffffffffb3422000
PAGE DIRECTORY: 23ffc5067
   PUD: 23ffc5000 => 1003b8067
   PMD: 1003b8000 => 1003b9067
   PTE: 1003b9000 => 800000005db2d121
  PAGE: 5db2d000

      PTE         PHYSICAL  FLAGS
800000005db2d121  5db2d000  (PRESENT|ACCESSED|GLOBAL|NX)

      PAGE        PHYSICAL      MAPPING       INDEX CNT FLAGS
ffffde8e4176cb40  5db2d000                0        0  1 fffffc0002000 reserved
```

5db2d000 idt_table idt_table pte crash idt_ta

```
crash> p &idt_table
$71 = (gate_desc (*)[256]) 0xffffffffb3f2d000 <idt_table>
crash> vtop 0xffffffffb3f2d000
VIRTUAL           PHYSICAL
ffffffffb3f2d000  5db2d000

PGD DIRECTORY: ffffffffb3422000
PAGE DIRECTORY: 5d027067
   PUD: 5d027ff0 => 5d028063
   PMD: 5d028cf8 => 1003ba063
   PTE: 1003ba968 => 800000005db2d121
  PAGE: 5db2d000

      PTE         PHYSICAL  FLAGS
800000005db2d121  5db2d000  (PRESENT|ACCESSED|GLOBAL|NX)
      PAGE        PHYSICAL      MAPPING       INDEX CNT FLAGS
ffffde8e4176cb40  5db2d000                0        0  1 fffffc0002000 reserve
```

idt_table idt_table CPU_ENTRY_AREA_RO_IDT_VADDR 0xfffffe000000000
idt idt_table CPU_ENTRY_AREA_RO_IDT_VADDR CPU_
only crash 1 x86 _PAGE_RW=0
load_idt 0xfffffe0000000000 idtr

```
load_idt(&idt_descr);

#define load_idt(dtr)    native_load_idt(dtr)
static __always_inline void native_load_idt(const struct desc_ptr *dtr)
{
 asm volatile("lidt %0"::"m" (*dtr));
}
```

idt_setup_apic_and_irq_gates set_memory_ro->change_page_attr_clear idt_table _PAGE_RW _

```
/* Make the IDT table read only */
set_memory_ro((unsigned long)&idt_table, 1);

int set_memory_ro(unsigned long addr, int numpages)
{
 return change_page_attr_clear(&addr, numpages, __pgprot(_PAGE_RW | _PAGE_DIRTY), 0);
}
```

x86-64

2

irq_entries_start SYM_CODE_START push jmp asm_common_
vector
asm_common_interrupt “ C ”

```
arch/x86/include/asm/idtentry.h

/* Device interrupts common/spurious */
DECLARE_IDTENTRY_IRQ(X86_TRAP_OTHER, common_interrupt);
```

DECLARE_IDTENTRY_IRQ .c .S

```
arch/x86/include/asm/idtentry.h

#ifndef __ASSEMBLER__
/**
 * DECLARE_IDTENTRY_ERRORCODE - Declare functions for simple IDT entry points
 *    Error code pushed by hardware
 * @vector: Vector number (ignored for C)
 * @func: Function name of the entry point
 *
 * Declares three functions:
 * - The ASM entry point: asm_##func
 * - The XEN PV trap entry point: xen_##func (maybe unused)
 * - The C handler called from the ASM entry point
 *
 * Same as DECLARE_IDTENTRY, but has an extra error_code argument for the
 * C-handler.
 */
#define DECLARE_IDTENTRY_ERRORCODE(vector, func)   \
 asmlinkage void asm_##func(void);    \
 asmlinkage void xen_asm_##func(void);    \
 __visible void func(struct pt_regs *regs, unsigned long error_code)

/**
 * DECLARE_IDTENTRY_IRQ - Declare functions for device interrupt IDT entry
 *     points (common/spurious)
 * @vector: Vector number (ignored for C)
 * @func: Function name of the entry point
 *
 * Maps to DECLARE_IDTENTRY_ERRORCODE()
 */
#define DECLARE_IDTENTRY_IRQ(vector, func)    \
 DECLARE_IDTENTRY_ERRORCODE(vector, func)
```

```
#else /* !__ASSEMBLER__ */
/* Entries for common/spurious (device) interrupts */
#define DECLARE_IDTENTRY_IRQ(vector, func)    \
 idtentry_irq vector func
```

c asm_common_interrupt .S idtentry_irq idtentry.h :

```
./arch/x86/kernel/idt.c:14:#include <asm/idtentry.h>
```

.c

```
./arch/x86/entry/entry_64.S:552:#include <asm/idtentry.h>
```

Linux “ C ” / stub CPU -> C handler-> ->iretq C handler +C C asm_common_interrupt

```
ffffffff81e00230:       f3 0f 1e fa             endbr64
ffffffff81e00234:       6a 20                   push   $0x20
ffffffff81e00236:       e9 05 13 00 00          jmp    ffffffff81e01540 <asm_common_interrup
ffffffff81e0023b:       cc                      int3
ffffffff81e0023c:       cc                      int3
ffffffff81e0023d:       cc                      int3
ffffffff81e0023e:       cc                      int3
ffffffff81e0023f:       cc                      int3
```

asm_common_interrupt arch/x86/entry/entry_64.S X86-64 X86-32 arch/x86/entry/entry_32.S

```
/* Device interrupts common/spurious */
DECLARE_IDTENTRY_IRQ(X86_TRAP_OTHER, common_interrupt);

/*
 * Dummy trap number so the low level ASM macro vector number checks do not
 * match which results in emitting plain IDTENTRY stubs without bells and
 * whistles.
 */
#define X86_TRAP_OTHER  0xFFFF
```

X86_TRAP_OTHER vector number irq_entries_start push

```
#define DECLARE_IDTENTRY_IRQ(vector, func)    \
 idtentry_irq vector func
```

idtentry_irq entry_64.S

```
arch/x86/entry/entry_64.S

/*
 * Interrupt entry/exit.
 *
 + The interrupt stubs push (vector) onto the stack, which is the error_code
 * position of idtentry exceptions, and jump to one of the two idtentry points
 * (common/spurious).
 *
 * common_interrupt is a hotpath, align it to a cache line
 */
.macro idtentry_irq vector cfunc
 .p2align CONFIG_X86_L1_CACHE_SHIFT
 idtentry \vector asm_\cfunc \cfunc has_error_code=1
.endm
```

.p2align 2^CONFIG_X86_L1_CACHE_SHIFT CONFIG_X86_L1_CACHE_SHIFT 6 64
L1 cache line cache line cache miss
asm_common_interrupt common_interrupt idtentry asm_common_interrupt common_in

```
arch/x86/entry/entry_64.S

/**
 * idtentry - Macro to generate entry stubs for simple IDT entries
 * @vector:  Vector number
 * @asmsym:  ASM symbol for the entry point
 * @cfunc:  C function to be called
 * @has_error_code: Hardware pushed error code on stack
 *
 * The macro emits code to set up the kernel context for straight forward
 * and simple IDT entries. No IST stack, no paranoid entry checks.
 */
.macro idtentry vector asmsym cfunc has_error_code:req
SYM_CODE_START(\asmsym)

 .if \vector == X86_TRAP_BP
  /* #BP advances %rip to the next instruction */
  UNWIND_HINT_IRET_ENTRY offset=\has_error_code*8 signal=0
 .else
  UNWIND_HINT_IRET_ENTRY offset=\has_error_code*8
 .endif

 ENDBR
 ASM_CLAC
 cld

 .if \has_error_code == 0
  pushq $-1   /* ORIG_RAX: no syscall to restart */
 .endif

 .if \vector == X86_TRAP_BP
```

```
  /*
   * If coming from kernel space, create a 6-word gap to allow the
   * int3 handler to emulate a call instruction.
   */
  testb $3, CS-ORIG_RAX(%rsp)
  jnz .Lfrom_usermode_no_gap_\@
  .rept 6
  pushq 5*8(%rsp)
  .endr
  UNWIND_HINT_IRET_REGS offset=8
.Lfrom_usermode_no_gap_\@:
 .endif

 idtentry_body \cfunc \has_error_code

_ASM_NOKPROBE(\asmsym)
SYM_CODE_END(\asmsym)
.endm
```

asm_common_interrupt

```
ffffffff81e01540 <asm_common_interrupt>:
ffffffff81e01540: f3 0f 1e fa          endbr64
ffffffff81e01544: 90                   nop
ffffffff81e01545: 90                   nop
ffffffff81e01546: 90                   nop
ffffffff81e01547: fc                   cld
ffffffff81e01548: e8 e3 06 00 00       call   ffffffff81e01c30 <error_entry>
ffffffff81e0154d: 48 89 c4             mov    %rax,%rsp
ffffffff81e01550: 48 89 e7             mov    %rsp,%rdi
ffffffff81e01553: 48 8b 74 24 78       mov    0x78(%rsp),%rsi
ffffffff81e01558: 48 c7 44 24 78 ff ff  movq   $0xffffffffffffffff,0x78(%rsp)
ffffffff81e0155f: ff ff
ffffffff81e01561: e8 5a 0c ef ff       call   ffffffff81cf21c0 <common_interrupt>
ffffffff81e01566: e9 05 08 00 00       jmp    ffffffff81e01d70 <error_return>
ffffffff81e0156b: 66 66 2e 0f 1f 84 00  data16 cs nopw 0x0(%rax,%rax,1)
ffffffff81e01572: 00 00 00 00
ffffffff81e01576: 66 2e 0f 1f 84 00 00  cs nopw 0x0(%rax,%rax,1)
ffffffff81e0157d: 00 00 00
```

SYM_CODE_START .global
if-else IDT ORC unwinder #BP breakpoint RIP
ENDBR ASM_CLAC ASM_CLAC .S .c .S .c

arch/x86/include/asm/smap.h

```
#ifdef __ASSEMBLER__
```

```
#define ASM_CLAC \
 ALTERNATIVE "", "clac", X86_FEATURE_SMAP
```

arch/x86/include/asm/alternative.h

```
/*
 * Issue one struct alt_instr descriptor entry (need to put it into
 * the section .altinstructions, see below). This entry contains
 * enough information for the alternatives patching code to patch an
 * instruction. See apply_alternatives().
 */
.macro altinstr_entry orig alt ft_flags orig_len alt_len
 .long \orig - .
 .long \alt - .
 .4byte \ft_flags
 .byte \orig_len
 .byte \alt_len
.endm

/*
 * Define an alternative between two instructions. If @feature is
 * present, early code in apply_alternatives() replaces @oldinstr with
 * @newinstr. ".skip" directive takes care of proper instruction padding
 * in case @newinstr is longer than @oldinstr.
 */
#define __ALTERNATIVE(oldinst, newinst, flag)    \
740:         \
 oldinst ;       \
741:         \
 .skip -(((744f-743f)-(741b-740b)) > 0) * ((744f-743f)-(741b-740b)),0x90 ;\
742:         \
 .pushsection .altinstructions,"a" ;    \
 altinstr_entry 740b,743f,flag,742b-740b,744f-743f ;  \
 .popsection ;       \
 .pushsection .altinstr_replacement,"ax" ;   \
743:         \
 newinst ;       \
744:         \
 .popsection ;

.macro ALTERNATIVE oldinstr, newinstr, ft_flags
 __ALTERNATIVE(\oldinstr, \newinstr, \ft_flags)
.endm
```

___ALTERNATIVE 740 (oldinst) 741 skip newinst oldinst nop(0x90)
743f) 743-744 (741b-740b) 741-742 b f b skip backward f skip skip forwar

feature flag + apply_alternatives CPU .altinstructions .altinstr_replacement “ ” “ ” nop patch
asm_common_interrupt X86_FEATURE_SMAP feature nop ASM_CLAC ALTERNA
Check CR4 SMAP(Supervisor Mode Access Prevention) clac
cld (DF, Direction Flag) DF EFLAGS DF=0 MOVS, CMPS, STOS, LODS, SCAS DF = 1 cld cli
error code stub push vector syscall number has_error_code orig_ax push -1 error code intel sdm vol3 6.3.1 error code vector 14 Page Fault error code vector 0 Divide Error error code intel sdm vol3 6.13 error code pushq -1 push error code / push procedure handler -1 handler intel sdm vol3 6.12.1

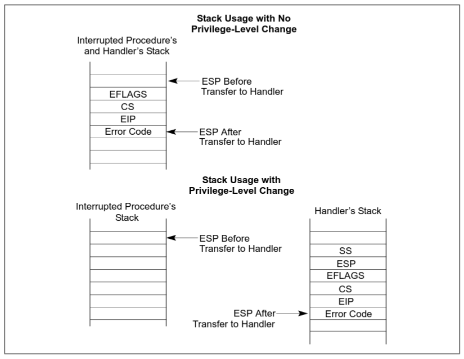

Stack Usage with No Privilege-Level Change
Interrupted Procedure's and Handler's Stack
EFLAGS
CS
EIP
Error Code
ESP Before Transfer to Handler
ESP After Transfer to Handler
Stack Usage with Privilege-Level Change
Interrupted Procedure's Stack
Handler's Stack
ESP Before Transfer to Handler
SS
ESP
EFLAGS
CS
EIP
Error Code
ESP After Transfer to Handler

Figure 2: /

int3 6 * 8 call int3 call testb cs cs rsp + 16 CS-ORIG_RAX offset 16

```
#define ORIG_RAX 120
#define CS 136
```

pt_regs oirg_rax error code syscall number pr_regs pt_regs -1

```
/*
 * C ABI says these regs are callee-preserved. They aren't saved on kernel entry
 * unless syscall needs a complete, fully filled "struct pt_regs".
 */
#define R15 0
#define R14 8
#define R13 16
#define R12 24
#define RBP 32
#define RBX 40
/* These regs are callee-clobbered. Always saved on kernel entry. */
#define R11 48
```

```
#define R10 56
#define R9 64
#define R8 72
#define RAX 80
#define RCX 88
#define RDX 96
#define RSI 104
#define RDI 112
/*
 * On syscall entry, this is syscall#. On CPU exception, this is error code.
 * On hw interrupt, it's IRQ number:
 */
#define ORIG_RAX 120
/* Return frame for iretq */
#define RIP 128
#define CS 136
#define EFLAGS 144
#define RSP 152
#define SS 160
#endif /* __ASSEMBLER__ */

/* top of stack page */
#define FRAME_SIZE 168

struct pt_regs {
 /*
  * C ABI says these regs are callee-preserved. They aren't saved on
  * kernel entry unless syscall needs a complete, fully filled
  * "struct pt_regs".
  */
 unsigned long r15;
 unsigned long r14;
 unsigned long r13;
 unsigned long r12;
 unsigned long bp;
 unsigned long bx;

 /* These regs are callee-clobbered. Always saved on kernel entry. */
 unsigned long r11;
 unsigned long r10;
 unsigned long r9;
 unsigned long r8;
 unsigned long ax;
 unsigned long cx;
 unsigned long dx;
 unsigned long si;
 unsigned long di;

 /*
  * orig_ax is used on entry for:
  * - the syscall number (syscall, sysenter, int80)
  * - error_code stored by the CPU on traps and exceptions
  * - the interrupt number for device interrupts
  *
  * A FRED stack frame starts here:
  *   1) It _always_ includes an error code;
  *
  *   2) The return frame for ERET[US] starts here, but
  *      the content of orig_ax is ignored.
  */
 unsigned long orig_ax;

 /* The IRETQ return frame starts here */
 unsigned long ip;

 union {
  /* CS selector */
  u16  cs;
  /* The extended 64-bit data slot containing CS */
  u64  csx;
  /* The FRED CS extension */
  struct fred_cs fred_cs;
 };

 unsigned long flags;
 unsigned long sp;

 union {
  /* SS selector */
  u16  ss;
  /* The extended 64-bit data slot containing SS */
  u64  ssx;
  /* The FRED SS extension */
  struct fred_ss fred_ss;
 };

 /*
  * Top of stack on IDT systems, while FRED systems have extra fields
  * defined above for storing exception related information, e.g. CR2 or
  * DR6.
  */
};
```

orig_ax orig_ax syscall number error code intel sdm vol3 error code 6.13 Table 6-1 orig_ax vector error code irq_entries_start push orig_ax error code identry cfunc common_interrupt has_error_code idtentry_body

```
/**
 * idtentry_body - Macro to emit code calling the C function
 * @cfunc:  C function to be called
 * @has_error_code: Hardware pushed error code on stack
 */
.macro idtentry_body cfunc has_error_code:req

 /*
  * Call error_entry() and switch to the task stack if from userspace.
  *
  * When in XENPV, it is already in the task stack, and it can't fault
  * for native_iret() nor native_load_gs_index() since XENPV uses its
  * own pvops for IRET and load_gs_index().  And it doesn't need to
  * switch the CR3.  So it can skip invoking error_entry().
  */
 ALTERNATIVE "call error_entry; movq %rax, %rsp", \
      "call xen_error_entry", X86_FEATURE_XENPV

 ENCODE_FRAME_POINTER
 UNWIND_HINT_REGS

 movq %rsp, %rdi   /* pt_regs pointer into 1st argument*/

 .if \has_error_code == 1
  movq ORIG_RAX(%rsp), %rsi /* get error code into 2nd argument*/
  movq $-1, ORIG_RAX(%rsp) /* no syscall to restart */
 .endif

 /* For some configurations \cfunc ends up being a noreturn. */
 ANNOTATE_REACHABLE
 call \cfunc

 jmp error_return
.endm
```

ALTERNATIVE Xen paravirtual guest call error_entry error_entry

```
arch/x86/entry/entry_64.S

SYM_CODE_START(error_entry)
 ANNOTATE_NOENDBR
 UNWIND_HINT_FUNC
 PUSH_AND_CLEAR_REGS save_ret=1
 ENCODE_FRAME_POINTER 8

 testb $3, CS+8(%rsp)
 jz .Lerror_kernelspace

 /*
  * We entered from user mode or we're pretending to have entered
  * from user mode due to an IRET fault.
  */
 swapgs
 FENCE_SWAPGS_USER_ENTRY
 /* We have user CR3.  Change to kernel CR3. */
 SWITCH_TO_KERNEL_CR3 scratch_reg=%rax
 IBRS_ENTER
 UNTRAIN_RET_FROM_CALL

 leaq 8(%rsp), %rdi   /* arg0 = pt_regs pointer */
 /* Put us onto the real thread stack. */
 jmp sync_regs

 /*
  * There are two places in the kernel that can potentially fault with
  * usergs. Handle them here.  B stepping K8s sometimes report a
  * truncated RIP for IRET exceptions returning to compat mode. Check
  * for these here too.
  */
.Lerror_kernelspace:
 leaq native_irq_return_iret(%rip), %rcx
 cmpq %rcx, RIP+8(%rsp)
 je .Lerror_bad_iret
 movl %ecx, %eax   /* zero extend */
 cmpq %rax, RIP+8(%rsp)
 je .Lbstep_iret
 cmpq $.Lgs_change, RIP+8(%rsp)
 jne .Lerror_entry_done_lfence

 /*
  * hack: .Lgs_change can fail with user gsbase.  If this happens, fix up
  * gsbase and proceed.  We'll fix up the exception and land in
  * .Lgs_change's error handler with kernel gsbase.
  */
 swapgs

 /*
```

```
  * Issue an LFENCE to prevent GS speculation, regardless of whether it is a
  * kernel or user gsbase.
  */
.Lerror_entry_done_lfence:
 FENCE_SWAPGS_KERNEL_ENTRY
 CALL_DEPTH_ACCOUNT
 leaq 8(%rsp), %rax   /* return pt_regs pointer */
 VALIDATE_UNRET_END
 RET

.Lbstep_iret:
 /* Fix truncated RIP */
 movq %rcx, RIP+8(%rsp)
 /* fall through */

.Lerror_bad_iret:
 /*
  * We came from an IRET to user mode, so we have user
  * gsbase and CR3.  Switch to kernel gsbase and CR3:
  */
 swapgs
 FENCE_SWAPGS_USER_ENTRY
 SWITCH_TO_KERNEL_CR3 scratch_reg=%rax
 IBRS_ENTER
 UNTRAIN_RET_FROM_CALL

 /*
  * Pretend that the exception came from user mode: set up pt_regs
  * as if we faulted immediately after IRET.
  */
 leaq 8(%rsp), %rdi   /* arg0 = pt_regs pointer */
 call fixup_bad_iret
 mov %rax, %rdi
 jmp sync_regs
SYM_CODE_END(error_entry)
```

PUSH_AND_CLEAR_REGS

```
arch/x86/entry/calling.h

.macro PUSH_AND_CLEAR_REGS rdx=%rdx rcx=%rcx rax=%rax save_ret=0 clear_bp=1 unwind_hint=1
 PUSH_REGS rdx=\rdx, rcx=\rcx, rax=\rax, save_ret=\save_ret unwind_hint=\unwind_hint
 CLEAR_REGS clear_bp=\clear_bp

/*
 * 64-bit system call stack frame layout defines and helpers,
 * for assembly code:
```

```
 */

.macro PUSH_REGS rdx=%rdx rcx=%rcx rax=%rax save_ret=0 unwind_hint=1
 .if \save_ret
 pushq %rsi  /* pt_regs->si */
 movq 8(%rsp), %rsi /* temporarily store the return address in %rsi */
 movq %rdi, 8(%rsp) /* pt_regs->di (overwriting original return address) */
 /* We just clobbered the return address - use the IRET frame for unwinding: */
 UNWIND_HINT_IRET_REGS offset=3*8
 .else
 pushq   %rdi  /* pt_regs->di */
 pushq   %rsi  /* pt_regs->si */
 .endif
 pushq \rdx  /* pt_regs->dx */
 pushq   \rcx  /* pt_regs->cx */
 pushq   \rax  /* pt_regs->ax */
 pushq   %r8  /* pt_regs->r8 */
 pushq   %r9  /* pt_regs->r9 */
 pushq   %r10  /* pt_regs->r10 */
 pushq   %r11  /* pt_regs->r11 */
 pushq %rbx  /* pt_regs->rbx */
 pushq %rbp  /* pt_regs->rbp */
 pushq %r12  /* pt_regs->r12 */
 pushq %r13  /* pt_regs->r13 */
 pushq %r14  /* pt_regs->r14 */
 pushq %r15  /* pt_regs->r15 */

 .if \unwind_hint
 UNWIND_HINT_REGS
 .endif

 .if \save_ret
 pushq %rsi  /* return address on top of stack */
 .endif
.endm
```

PUSH_REGS pushq error_entry 15 rsp rsi rsi rsp+8 push rsi error_entry mov rsp error_entry rsi rsi rsp 8 rsi rsi mov rdi error_entry error_entry rsi rsp+8 CLEAR_REGS 15

```
.macro CLEAR_REGS clear_bp=1
 /*
  * Sanitize registers of values that a speculation attack might
  * otherwise want to exploit. The lower registers are likely clobbered
  * well before they could be put to use in a speculative execution
  * gadget.
  */
 xorl %esi,  %esi /* nospec si  */
 xorl %edx,  %edx /* nospec dx  */
 xorl %ecx,  %ecx /* nospec cx  */
 xorl %r8d,  %r8d /* nospec r8  */
 xorl %r9d,  %r9d /* nospec r9  */
 xorl %r10d, %r10d /* nospec r10 */
 xorl %r11d, %r11d /* nospec r11 */
 xorl %ebx,  %ebx /* nospec rbx */
 .if \clear_bp
 xorl %ebp,  %ebp /* nospec rbp */
 .endif
 xorl %r12d, %r12d /* nospec r12 */
 xorl %r13d, %r13d /* nospec r13 */
 xorl %r14d, %r14d /* nospec r14 */
 xorl %r15d, %r15d /* nospec r15 */

.endm
```

error_entry CS+8 code segment register testb 3 code segment register AND Zero Flags testb ZF = 1 code segment register 0 / cpu jz error_kernelspace / cpu code segment register 11

swapgs gs MSR C0000102H IA32_KERNEL_GS_BASE x86-64 gs.base TLS GS.base per-cpu per-CPU swapgs swapgs gs.base “ ” per-cpu gs per-CPU swapgs gs.base tls

FENCE_SWAPGS_USER_ENTRY

```
.macro FENCE_SWAPGS_USER_ENTRY
 ALTERNATIVE "", "lfence", X86_FEATURE_FENCE_SWAPGS_USER
.endm
```

nop lfence 3 nop lfence memory fence load fence store A B” LFENCE CPU Producer flag=1 Consumer flag=1 flag=1 LFENCE consumer flag=1 consumer LFENCE flag cpu

```
ffffffff81001c8d:       65 48 8b 15 1b 31 c5    mov    %gs:0x2c5311b(%rip),%rdx        # fff
```

swapgs mov swapgs SWITCH_TO_KERNEL_CR3

```
.macro SWITCH_TO_KERNEL_CR3 scratch_reg:req
 ALTERNATIVE "jmp .Lend_\@", "", X86_FEATURE_PTI
 mov %cr3, \scratch_reg
 ADJUST_KERNEL_CR3 \scratch_reg
 mov \scratch_reg, %cr3
```

```
.Lend_\@:
.endm
```

PTI / CONFIG_MITIGATION_PAGE_TABLE_ISOLATION ALTERNATIVE

```
.macro ADJUST_KERNEL_CR3 reg:req
 ALTERNATIVE "", "SET_NOFLUSH_BIT \reg", X86_FEATURE_PCID
 /* Clear PCID and "MITIGATION_PAGE_TABLE_ISOLATION bit", point CR3 at kernel pagetables: */
 andq    $(~PTI_USER_PGTABLE_AND_PCID_MASK), \reg
.endm

.macro SET_NOFLUSH_BIT reg:req
 bts $X86_CR3_PCID_NOFLUSH_BIT, \reg
.endm

#define X86_CR3_PCID_NOFLUSH_BIT 63 /* Preserve old PCID */
```

pcid feature Intel intel sdm vol3 4.10.1 TLB tlb tlb cr3
63 Intel “no flush” CR3.NOFLUSH mov cr3 tlb intel sdm vol3
4.10.4.1 CR4.PCIDE X86_FEATURE_PCID feature
CR4.PCIDE = 0 PCID = 000H TLB Global Pages PCID
= 000H paging-structure caches / PCID CR4.PCIDE=0 CPU PCID=0 CR3 PC

CR4.PCIDE = 1 MOV CR3 63 0 12 bits 11:0 PCID TL-
B PCID CPU PCID PCID CR3 PCID 12 PCID TLB
63=0 CR3 PCID TLB PCID TLB
CR4.PCIDE = 1 63 1 TLB
invlpg/invpcid tlb intel sdm vol3 4.10.4.1
cpu pcide PTI feature X86_FEATURE_PTI ADJUST_KERNEL_CR3 bts CR3
63 1 mov cr3 CR4.PCIDE = 1 CR3.bit63 = 1 TLB
ADJUST_KERNEL_CR3 andq PTI_USER_PGTABLE_AND_PCID_MASK

```
/*
 * MITIGATION_PAGE_TABLE_ISOLATION PGDs are 8k.  Flip bit 12 to switch between the two
 * halves:
 */
#define PTI_USER_PGTABLE_BIT  PAGE_SHIFT
#define PTI_USER_PGTABLE_MASK  (1 << PTI_USER_PGTABLE_BIT)
#define PTI_USER_PCID_BIT  X86_CR3_PTI_PCID_USER_BIT
#define PTI_USER_PCID_MASK  (1 << PTI_USER_PCID_BIT)
#define PTI_USER_PGTABLE_AND_PCID_MASK  (PTI_USER_PCID_MASK | PTI_USER_PGTABLE_MASK)

#ifdef CONFIG_MITIGATION_PAGE_TABLE_ISOLATION
# define X86_CR3_PTI_PCID_USER_BIT 11
#endif
```

X86_CR3_PTI_PCID_USER_BIT pcid pcid 12 user/kernel pcid 1 pcid 0 pcid
2048 PCID bit11=1 2048

PAGE_SHIFT 12 bit 12 4KB 8KB 8KB MITIGATION_PAGE_TAB
11 bit 12 ADJUST_KERNEL_CR3 0xffffffffffffe7ff bit 11 bit 12 0 cr3 8
pcid asic pcid 4096 pcid 6

```
#define TLB_NR_DYN_ASIDS 6
```

CPU Linux 6 mm_struct ASID task 6 struct
tlb_state per-cpu ASID cache line

```
static inline u16 user_pcid(u16 asid)
{
 u16 ret = kern_pcid(asid);
#ifdef CONFIG_MITIGATION_PAGE_TABLE_ISOLATION
 ret |= 1 << X86_CR3_PTI_PCID_USER_BIT;
#endif
 return ret;
}

/*
 * Given @asid, compute kPCID
 */
static inline u16 kern_pcid(u16 asid)
{
 VM_WARN_ON_ONCE(asid > MAX_ASID_AVAILABLE);

#ifdef CONFIG_MITIGATION_PAGE_TABLE_ISOLATION
 /*
  * Make sure that the dynamic ASID space does not conflict with the
  * bit we are using to switch between user and kernel ASIDs.
  */
 BUILD_BUG_ON(TLB_NR_DYN_ASIDS >= (1 << X86_CR3_PTI_PCID_USER_BIT));

 /*
  * The ASID being passed in here should have respected the
  * MAX_ASID_AVAILABLE and thus never have the switch bit set.
  */
 VM_WARN_ON_ONCE(asid & (1 << X86_CR3_PTI_PCID_USER_BIT));
#endif
 /*
  * The dynamically-assigned ASIDs that get passed in are small
  * (<TLB_NR_DYN_ASIDS).  They never have the high switch bit set,
  * so do not bother to clear it.
  *
  * If PCID is on, ASID-aware code paths put the ASID+1 into the
  * PCID bits.  This serves two purposes.  It prevents a nasty
  * situation in which PCID-unaware code saves CR3, loads some other
  * value (with PCID == 0), and then restores CR3, thus corrupting
  * the TLB for ASID 0 if the saved ASID was nonzero.  It also means
  * that any bugs involving loading a PCID-enabled CR3 with
  * CR4.PCIDE off will trigger deterministically.
  */
 return asid + 1;
}
```

kern_pcid(asid) asid+1 ASID=0 user_pcid(asid) kern_pcid(asid) PTI PCID U
flush PTI
pcid asid+1 asid=0 pcid=0 pcid-unware
user_pcid native_flush_tlb_one_user

```
STATIC_NOPV void native_flush_tlb_one_user(unsigned long addr)
{
 u32 loaded_mm_asid;
 bool cpu_pcide;

 /* Flush 'addr' from the kernel PCID: */
 invlpg(addr);

 /* If PTI is off there is no user PCID and nothing to flush. */
 if (!static_cpu_has(X86_FEATURE_PTI))
  return;

 loaded_mm_asid = this_cpu_read(cpu_tlbstate.loaded_mm_asid);
 cpu_pcide      = this_cpu_read(cpu_tlbstate.cr4) & X86_CR4_PCIDE;

 /*
  * invpcid_flush_one(pcid>0) will #GP if CR4.PCIDE==0.  Check
  * 'cpu_pcide' to ensure that *this* CPU will not trigger those
  * #GP's even if called before CR4.PCIDE has been initialized.
  */
 if (boot_cpu_has(X86_FEATURE_INVPCID) && cpu_pcide)
  invpcid_flush_one(user_pcid(loaded_mm_asid), addr);
 else
  invalidate_user_asid(loaded_mm_asid);
}
```

per-cpu cpu_tlbstate.loaded_mm_asid switch_mm_irqs_off mm_struct

```
this_cpu_write(cpu_tlbstate.loaded_mm_asid, ns.asid);
```

```
ns = choose_new_asid(next, next_tlb_gen);
```

```
this_cpu_write(cpu_tlbstate.loaded_mm_asid, ns.asid);
```

```
static struct new_asid choose_new_asid(struct mm_struct *next, u64 next_tlb_gen)
{
 struct new_asid ns;
 u16 asid;

 if (!static_cpu_has(X86_FEATURE_PCID)) {
  ns.asid = 0;
  ns.need_flush = 1;
  return ns;
 }

 /*
  * TLB consistency for global ASIDs is maintained with hardware assisted
  * remote TLB flushing. Global ASIDs are always up to date.
  */
 if (cpu_feature_enabled(X86_FEATURE_INVLPGB)) {
  u16 global_asid = mm_global_asid(next);

  if (global_asid) {
   ns.asid = global_asid;
   ns.need_flush = 0;
   return ns;
  }
 }

 if (this_cpu_read(cpu_tlbstate.invalidate_other))
  clear_asid_other();

 for (asid = 0; asid < TLB_NR_DYN_ASIDS; asid++) {
  if (this_cpu_read(cpu_tlbstate.ctxs[asid].ctx_id) !=
      next->context.ctx_id)
   continue;

  ns.asid = asid;
  ns.need_flush = (this_cpu_read(cpu_tlbstate.ctxs[asid].tlb_gen) < next_tlb_gen);
  return ns;
 }

 /*
  * We don't currently own an ASID slot on this CPU.
  * Allocate a slot.
  */
 ns.asid = this_cpu_add_return(cpu_tlbstate.next_asid, 1) - 1;
 if (ns.asid >= TLB_NR_DYN_ASIDS) {
  ns.asid = 0;
  this_cpu_write(cpu_tlbstate.next_asid, 1);
 }
 ns.need_flush = true;

 return ns;
}
```

asid pcid switch_mm_irqs_off->load_new_mm_cr3->write_cr3 cr3 tlb_state per-cpu cpu tlb tlb_state::loaded_mm_asid tlb_state::ctxs type: tlb_context tlb_context::c
mm_struct mm mm tlb
copy_from_user()/copy_to_user() pcid 0
>PA PCID=0 TLB B PCID=0 TLB A B TLB
PCID Linux PCID user_pcid = base_pcid | (1 «11) ernel_pcid =
base_pcid base_pcid 5
cr3 pcid tlb CONFIG_MITIGATION_PAGE_TABLE_ISOLATION
fault PTE CPL=0 (CPL=0) PTI Meltdown CPU
Intel (out-of-order execution) CPU speculatively (cache)
pcid/asid error_entry IBRS_ENTER

```
/*
 * IBRS kernel mitigation for Spectre_v2.
 *
 * Assumes full context is established (PUSH_REGS, CR3 and GS) and it clobbers
 * the regs it uses (AX, CX, DX). Must be called before the first RET
 * instruction (NOTE! UNTRAIN_RET includes a RET instruction)
 *
 * The optional argument is used to save/restore the current value,
 * which is used on the paranoid paths.
 *
 * Assumes x86_spec_ctrl_{base,current} to have SPEC_CTRL_IBRS set.
 */
.macro IBRS_ENTER save_reg
#ifdef CONFIG_MITIGATION_IBRS_ENTRY
 ALTERNATIVE "jmp .Lend_\@", "", X86_FEATURE_KERNEL_IBRS
 movl $MSR_IA32_SPEC_CTRL, %ecx

.ifnb \save_reg
 rdmsr
 shl $32, %rdx
 or %rdx, %rax
 mov %rax, \save_reg
 test $SPEC_CTRL_IBRS, %eax
 jz .Ldo_wrmsr_\@
 lfence
 jmp .Lend_\@
.Ldo_wrmsr_\@:
.endif
```

```
 movq PER_CPU_VAR(x86_spec_ctrl_current), %rdx
 movl %edx, %eax
 shr $32, %rdx
 wrmsr
.Lend_\@:
#endif
.endm
```

MSR_IA32_SPEC_CTRL 0x48 intel sdm vol4 Register Address 0x48 msr Speculation Control Bit 0 IBRS Indirect Branch Restricted Speculation 1 CPU Spectre v2 / wrmsr CPU Enhanced IBRS IBRS MSR
wrmsr msr ecx edx:eax 32 64
UNTRAIN_RET_FROM_CALL
error_entry

```
leaq 8(%rsp), %rdi   /* arg0 = pt_regs pointer */
/* Put us onto the real thread stack. */
jmp sync_regs
```

rsp 8 pt_regs rsp error_entry asm_common_interrupt call error_entry mov ffffffff81e0154d +8 r15

```
ffffffff81e01540 <asm_common_interrupt>:
ffffffff81e01540: f3 0f 1e fa          endbr64
ffffffff81e01544: 90                   nop
ffffffff81e01545: 90                   nop
ffffffff81e01546: 90                   nop
ffffffff81e01547: fc                   cld
ffffffff81e01548: e8 e3 06 00 00       call   ffffffff81e01c30 <error_entry>
ffffffff81e0154d: 48 89 c4             mov    %rax,%rsp
ffffffff81e01550: 48 89 e7             mov    %rsp,%rdi
ffffffff81e01553: 48 8b 74 24 78       mov    0x78(%rsp),%rsi
ffffffff81e01558: 48 c7 44 24 78 ff ff  movq   $0xffffffffffffffff,0x78(%rsp)
ffffffff81e0155f: ff ff
ffffffff81e01561: e8 5a 0c ef ff       call   ffffffff81cf21c0 <common_interrupt>
ffffffff81e01566: e9 05 08 00 00       jmp    ffffffff81e01d70 <error_return>
ffffffff81e0156b: 66 66 2e 0f 1f 84 00  data16 cs nopw 0x0(%rax,%rax,1)
ffffffff81e01572: 00 00 00 00
ffffffff81e01576: 66 2e 0f 1f 84 00 00  cs nopw 0x0(%rax,%rax,1)
ffffffff81e0157d: 00 00 00
```

jmp sync_regs sync_regs ret asm_common_interrupt sync_regs

```
/*
 * Help handler running on a per-cpu (IST or entry trampoline) stack
 * to switch to the normal thread stack if the interrupted code was in
 * user mode. The actual stack switch is done in entry_64.S
 */
asmlinkage __visible noinstr struct pt_regs *sync_regs(struct pt_regs *eregs)
{
 struct pt_regs *regs = (struct pt_regs *)current_top_of_stack() - 1;
 if (regs != eregs)
  *regs = *eregs;
 return regs;
}
```

current_top_of_stack task_struct::stack
error_entry error_entry
sync_regs error_entry rax pt_regs / asm_common_interrupt idtentry_bo

```
/**
 * idtentry_body - Macro to emit code calling the C function
 * @cfunc:  C function to be called
 * @has_error_code: Hardware pushed error code on stack
 */
.macro idtentry_body cfunc has_error_code:req

 /*
  * Call error_entry() and switch to the task stack if from userspace.
  *
  * When in XENPV, it is already in the task stack, and it can't fault
  * for native_iret() nor native_load_gs_index() since XENPV uses its
  * own pvops for IRET and load_gs_index().  And it doesn't need to
  * switch the CR3.  So it can skip invoking error_entry().
  */
 ALTERNATIVE "call error_entry; movq %rax, %rsp", \
      "call xen_error_entry", X86_FEATURE_XENPV

 ENCODE_FRAME_POINTER
 UNWIND_HINT_REGS

 movq %rsp, %rdi   /* pt_regs pointer into 1st argument*/

 .if \has_error_code == 1
  movq ORIG_RAX(%rsp), %rsi /* get error code into 2nd argument*/
  movq $-1, ORIG_RAX(%rsp) /* no syscall to restart */
 .endif

 /* For some configurations \cfunc ends up being a noreturn. */
 ANNOTATE_REACHABLE
 call \cfunc

 jmp error_return
.endm
```

error_entry->sync_regs ret idtentry_body rax pt_regs mov %rsp, %rdi rdi rsi pt_regs orig_rax irq_entries_start push vector cfunc idtentr >idtentry->idtentry_body common_interrupt call c “ ” common_interrupt

```
/*
 * common_interrupt() handles all normal device IRQ's (the special SMP
 * cross-CPU interrupts have their own entry points).
 */
DEFINE_IDTENTRY_IRQ(common_interrupt)
{
 struct pt_regs *old_regs = set_irq_regs(regs);

 /* entry code tells RCU that we're not quiescent.  Check it. */
 RCU_LOCKDEP_WARN(!rcu_is_watching(), "IRQ failed to wake up RCU");

 if (unlikely(!call_irq_handler(vector, regs)))
  apic_eoi();

 set_irq_regs(old_regs);
}
```

common_interrupt DEFINE_IDTENTRY_IRQ irq

```
DEFINE_IDTENTRY_IRQ(spurious_interrupt)
```

DEFINE_IDTENTRY_IRQ

```
/**
 * DEFINE_IDTENTRY_IRQ - Emit code for device interrupt IDT entry points
 * @func: Function name of the entry point
 *
 * The vector number is pushed by the low level entry stub and handed
 * to the function as error_code argument which needs to be truncated
 * to an u8 because the push is sign extending.
 *
 * irq_enter/exit_rcu() are invoked before the function body and the
 * KVM L1D flush request is set. Stack switching to the interrupt stack
 * has to be done in the function body if necessary.
 */
#define DEFINE_IDTENTRY_IRQ(func)     \
static void __##func(struct pt_regs *regs, u32 vector);   \
        \
__visible noinstr void func(struct pt_regs *regs,   \
       unsigned long error_code)   \
{         \
 irqentry_state_t state = irqentry_enter(regs);   \
 u32 vector = (u32)(u8)error_code;    \
         \
 kvm_set_cpu_l1tf_flush_l1d();                                   \
 instrumentation_begin();     \
 run_irq_on_irqstack_cond(__##func, regs, vector);  \
 instrumentation_end();      \
 irqentry_exit(regs, state);     \
}         \
         \
static noinline void __##func(struct pt_regs *regs, u32 vector)
```

common_interrupt

```
static void __common_interrupt(struct pt_regs *regs, u32 vector);

__visible noinstr void common_interrupt(struct pt_regs *regs,
       unsigned long error_code)
{
 irqentry_state_t state = irqentry_enter(regs);
 u32 vector = (u32)(u8)error_code;

 kvm_set_cpu_l1tf_flush_l1d();
 instrumentation_begin();
 run_irq_on_irqstack_cond(__common_interrupt, regs, vector);
 instrumentation_end();
 irqentry_exit(regs, state);
}

static noinline void __common_interrupt(struct pt_regs *regs, u32 vector)
{
 struct pt_regs *old_regs = set_irq_regs(regs);

 /* entry code tells RCU that we're not quiescent.  Check it. */
 RCU_LOCKDEP_WARN(!rcu_is_watching(), "IRQ failed to wake up RCU");

 if (unlikely(!call_irq_handler(vector, regs)))
  apic_eoi();

 set_irq_regs(old_regs);
}
```

irqentry_enter/irqentry_exit irqentry RCU lockdep
error_code vector u8 u32 vector
kvm_set_cpu_l1tf_flush_l1d guest

```
arch/x86/include/asm/hardirq.h

static __always_inline void kvm_set_cpu_l1tf_flush_l1d(void)
```

```
{
 __this_cpu_write(irq_stat.kvm_cpu_l1tf_flush_l1d, 1);
}
```

guest vmx_l1d_flush l1d cache
instrumentation_begin/instrumentation_end CONFIG_NOINSTR_VALIDATION
run_irq_on_irqstack_cond

```
arch/x86/include/asm/irq_stack.h

#define run_irq_on_irqstack_cond(func, regs, vector)   \
{         \
 assert_function_type(func, void (*)(struct pt_regs *, u32)); \
 assert_arg_type(regs, struct pt_regs *);   \
 assert_arg_type(vector, u32);     \
         \
 call_on_irqstack_cond(func, regs, ASM_CALL_IRQ,   \
         IRQ_CONSTRAINTS, regs, vector);  \
}
```

```
arch/x86/include/asm/irq_stack.h

/*
 * Macro to invoke system vector and device interrupt C handlers.
 */
#define call_on_irqstack_cond(func, regs, asm_call, constr, c_args...) \
{         \
 /*        \
  * User mode entry and interrupt on the irq stack do not \
  * switch stacks. If from user mode the task stack is empty. \
  */        \
 if (user_mode(regs) || __this_cpu_read(hardirq_stack_inuse)) { \
  irq_enter_rcu();     \
  func(c_args);      \
  irq_exit_rcu();      \
 } else {       \
  /*       \
   * Mark the irq stack inuse _before_ and unmark _after_ \
   * switching stacks. Interrupts are disabled in both \
   * places. Invoke the stack switch macro with the call \
   * sequence which matches the above direct invocation. \
   */       \
  __this_cpu_write(hardirq_stack_inuse, true);  \
  call_on_irqstack(func, asm_call, constr);  \
  __this_cpu_write(hardirq_stack_inuse, false);  \
 }       \
}
```

common_interrupt irq_entries_start/error_entry irq_ent

```
        IRQ

CPU
  -    TSS.sp0  IST        ←  "  "
  -   SS, RSP, RFLAGS, CS, RIP
  -   IDT

entry_64.S

        eregs

     sync_regs(eregs)

         eregs →   task
          regs


      **  **

call_on_irqstack_cond(func, regs, ...)
    user_mode(regs) == true
         func()

func() =        handle_irq_event
  task

irq_exit_rcu() →

iret →
```

rsp task CPU call_on_irqstack_cond else task

```
      task

       CPL=0
   → CPU

      (entry_64.S)
         task
```

```
     call_on_irqstack_cond(func, regs, ...)
       user_mode(regs) == false
       hardirq_stack_inuse == false
           per-CPU hardirq
           call_on_irqstack(func, asm_call, ...)

   __this_cpu_read(hardirq_stack_ptr)
     →   map_irq_stack()    per-CPU

func() =     handle_irq_event
  per-CPU **hardirq  **


  __this_cpu_write(hardirq_stack_inuse, false)
  irq_exit_rcu()

iret
```

hardirq_stack_inuse true CPU hardirq hardirq_stack_inuse

```
   hardirq


   → CPU     CPL=0

   entry_64.S

   call_on_irqstack_cond(func, regs, ...)
       user_mode(regs) == false
       hardirq_stack_inuse == true

         func()    hardirq

func() =   IRQ handler
    CPU   **  hardirq  **

irq_exit_rcu() →
```

"CPU TSS.IST TSS.sp0 " sp0 tss task-state segment 64 intel sdm vol3 Figure 7-11 rsp0 rsp1 rsp2 7 IST Intel sdm vol3 call/interrupt/exception

```
When the processor performs a call to the exception- or interrupt-handler procedure: If the
a. The segment selector and stack pointer for the stack to be used by the handler are obtai
for the currently executing task. On this new stack, the processor pushes the stack segment
stack pointer of the interrupted procedure.
b. The processor then saves the current state of the EFLAGS, CS, and EIP registers on the ne
```

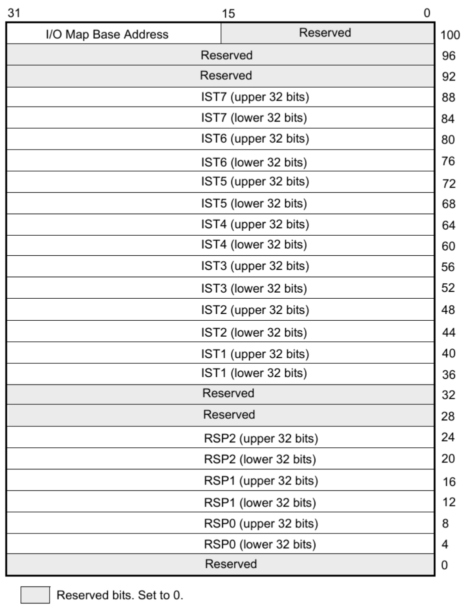


Figure 3: 64bit tss

```
Figure 6-4).
c. If an exception causes an error code to be saved, it is pushed on the new stack after the
```

cpu user mode 3 0 tss tss rsp Intel
sdm vol3

```
If the call requires a change in privilege level, the processor also switches to the stack f
```

rsp level 0 rsp0
Linux x86_hw_tss x86-64

```
arch/x86/include/asm/processor.h

struct x86_hw_tss {
 u32   reserved1;
 u64   sp0;
 u64   sp1;

 /*
  * Since Linux does not use ring 2, the 'sp2' slot is unused by
  * hardware.  entry_SYSCALL_64 uses it as scratch space to stash
  * the user RSP value.
  */
 u64   sp2;
 u64   reserved2;
 u64   ist[7];
 u32   reserved3;
 u32   reserved4;
 u16   reserved5;
 u16   io_bitmap_base;

} __attribute__((packed));
```

cpu tss tss base limit x86-64 tss

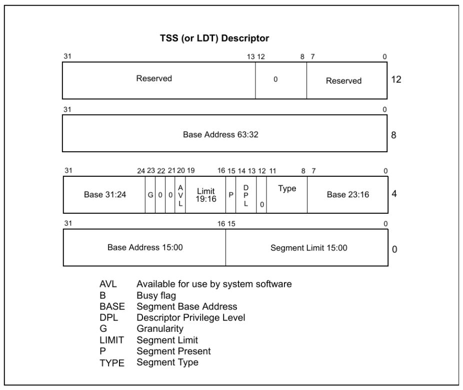


Figure 4: 64 tss/ldt

```
arch/x86/include/asm/desc_defs.h

/* LDT or TSS descriptor in the GDT. */
struct ldttss_desc {
 u16 limit0;
 u16 base0;

 u16 base1 : 8, type : 5, dpl : 2, p : 1;
 u16 limit1 : 4, zero0 : 3, g : 1, base2 : 8;
#ifdef CONFIG_X86_64
 u32 base3;
 u32 zero1;
#endif
} __attribute__((packed));
```

x86-64 tss sp0 sp1 sp2 tss X86 cpu 16 real mode 32 protected mode x86 cpu 64 IA-32e mode 64-bit mode Compatibility 64 32 app

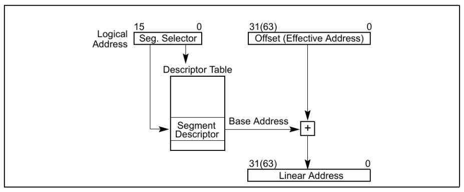


Figure 5:

intel sdm vol3
GDT Global Descriptor Table LDT Local Descriptor Table
GDTR Global Descriptor Table Register GDT LDTR Local Descrip-
torTable Register LDT CS SS DS ES FS GS GDT LDT

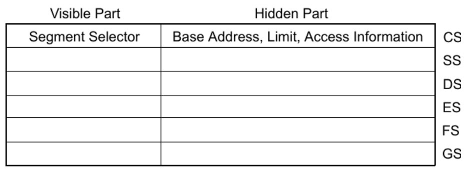


Figure 6:

gdt/ldt

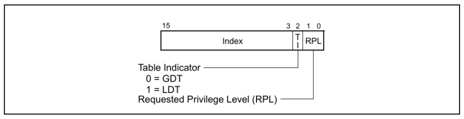


Figure 7:

bit 3-15 gdt/ldt TI gdt/ldt
task register x86 tss task register tss gdt gdt tss

x86 tss tss rspN gdt lgdt gdt tss task
register tss gdt task register tss TR
Linux gdt Linux cpu gdt page

`arch/x86/include/asm/cpu_entry_area.h`

```
/*
 * cpu_entry_area is a percpu region that contains things needed by the CPU
 * and early entry/exit code.  Real types aren't used for all fields here
```

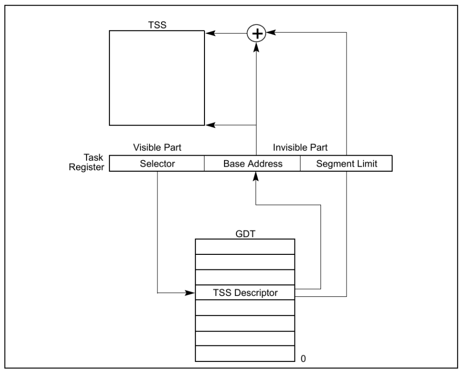


Figure 8: task register tss

```
 * to avoid circular header dependencies.
 *
 * Every field is a virtual alias of some other allocated backing store.
 * There is no direct allocation of a struct cpu_entry_area.
 */


struct cpu_entry_area {
 char gdt[PAGE_SIZE];
      ...
}
```

setup_cpu_entry_area gdt page

```
arch/x86/mm/cpu_entry_area.c

/* Setup the fixmap mappings only once per-processor */
static void __init setup_cpu_entry_area(unsigned int cpu)
{
      struct cpu_entry_area *cea = get_cpu_entry_area(cpu);
#ifdef CONFIG_X86_64
 /* On 64-bit systems, we use a read-only fixmap GDT and TSS. */
 pgprot_t gdt_prot = PAGE_KERNEL_RO;
 pgprot_t tss_prot = PAGE_KERNEL_RO;
#else
 /*
  * On 32-bit systems, the GDT cannot be read-only because
  * our double fault handler uses a task gate, and entering through
  * a task gate needs to change an available TSS to busy.  If the
  * GDT is read-only, that will triple fault.  The TSS cannot be
  * read-only because the CPU writes to it on task switches.
  */
 pgprot_t gdt_prot = PAGE_KERNEL;
 pgprot_t tss_prot = PAGE_KERNEL;
#endif
      ...
 cea_set_pte(&cea->gdt, get_cpu_gdt_paddr(cpu), gdt_prot);
      ...
}
```

cea_set_pte cpu_entry_area alloc_* percpu cea_set_pte cpu_entry

```
arch/x86/include/asm/desc.h

/* Provide the physical address of the GDT page. */
static inline phys_addr_t get_cpu_gdt_paddr(unsigned int cpu)
{
 return per_cpu_ptr_to_phys(get_cpu_gdt_rw(cpu));
}

static inline struct desc_struct *get_cpu_gdt_rw(unsigned int cpu)
{
 return per_cpu(gdt_page, cpu).gdt;
}
```

cpu_entry_area backingstore gdt_page per_cpu percpu DEFINE

```
arch/x86/kernel/cpu/common.c

DEFINE_PER_CPU_PAGE_ALIGNED(struct gdt_page, gdt_page) = { .gdt = {
#ifdef CONFIG_X86_64
 /*
  * We need valid kernel segments for data and code in long mode too
  * IRET will check the segment types  kkeil 2000/10/28
  * Also sysret mandates a special GDT layout
  *
  * TLS descriptors are currently at a different place compared to i386.
  * Hopefully nobody expects them at a fixed place (Wine?)
  */
 [GDT_ENTRY_KERNEL32_CS]  = GDT_ENTRY_INIT(DESC_CODE32, 0, 0xfffff),
 [GDT_ENTRY_KERNEL_CS]  = GDT_ENTRY_INIT(DESC_CODE64, 0, 0xfffff),
 [GDT_ENTRY_KERNEL_DS]  = GDT_ENTRY_INIT(DESC_DATA64, 0, 0xfffff),
 [GDT_ENTRY_DEFAULT_USER32_CS] = GDT_ENTRY_INIT(DESC_CODE32 | DESC_USER, 0, 0xfffff),
 [GDT_ENTRY_DEFAULT_USER_DS] = GDT_ENTRY_INIT(DESC_DATA64 | DESC_USER, 0, 0xfffff),
 [GDT_ENTRY_DEFAULT_USER_CS] = GDT_ENTRY_INIT(DESC_CODE64 | DESC_USER, 0, 0xfffff),
#else
 [GDT_ENTRY_KERNEL_CS]  = GDT_ENTRY_INIT(DESC_CODE32, 0, 0xfffff),
 [GDT_ENTRY_KERNEL_DS]  = GDT_ENTRY_INIT(DESC_DATA32, 0, 0xfffff),
 [GDT_ENTRY_DEFAULT_USER_CS] = GDT_ENTRY_INIT(DESC_CODE32 | DESC_USER, 0, 0xfffff),
 [GDT_ENTRY_DEFAULT_USER_DS] = GDT_ENTRY_INIT(DESC_DATA32 | DESC_USER, 0, 0xfffff),
 /*
  * Segments used for calling PnP BIOS have byte granularity.
  * They code segments and data segments have fixed 64k limits,
  * the transfer segment sizes are set at run time.
  */
 [GDT_ENTRY_PNPBIOS_CS32] = GDT_ENTRY_INIT(DESC_CODE32_BIOS, 0, 0xffff),
 [GDT_ENTRY_PNPBIOS_CS16] = GDT_ENTRY_INIT(DESC_CODE16, 0, 0xffff),
 [GDT_ENTRY_PNPBIOS_DS]  = GDT_ENTRY_INIT(DESC_DATA16, 0, 0xffff),
 [GDT_ENTRY_PNPBIOS_TS1]  = GDT_ENTRY_INIT(DESC_DATA16, 0, 0),
 [GDT_ENTRY_PNPBIOS_TS2]  = GDT_ENTRY_INIT(DESC_DATA16, 0, 0),
 /*
  * The APM segments have byte granularity and their bases
  * are set at run time.  All have 64k limits.
  */
 [GDT_ENTRY_APMBIOS_BASE] = GDT_ENTRY_INIT(DESC_CODE32_BIOS, 0, 0xffff),
 [GDT_ENTRY_APMBIOS_BASE+1] = GDT_ENTRY_INIT(DESC_CODE16, 0, 0xffff),
 [GDT_ENTRY_APMBIOS_BASE+2] = GDT_ENTRY_INIT(DESC_DATA32_BIOS, 0, 0xffff),

 [GDT_ENTRY_ESPFIX_SS]  = GDT_ENTRY_INIT(DESC_DATA32, 0, 0xfffff),
 [GDT_ENTRY_PERCPU]  = GDT_ENTRY_INIT(DESC_DATA32, 0, 0xfffff),
#endif
} };


include/linux/percpu-defs.h

#define DEFINE_PER_CPU_PAGE_ALIGNED(type, name)    \
 DEFINE_PER_CPU_SECTION(type, name, "..page_aligned")  \
 __aligned(PAGE_SIZE)

#define DEFINE_PER_CPU_SECTION(type, name, sec)    \
 __PCPU_ATTRS(sec) __typeof__(type) name

/*
 * Base implementations of per-CPU variable declarations and definitions, where
 * the section in which the variable is to be placed is provided by the
 * 'sec' argument.  This may be used to affect the parameters governing the
 * variable's storage.
 *
 * NOTE!  The sections for the DECLARE and for the DEFINE must match, lest
 * linkage errors occur due the compiler generating the wrong code to access
 * that section.
 */
#define __PCPU_ATTRS(sec)      \
 __percpu __attribute__((section(PER_CPU_BASE_SECTION sec))) \
 PER_CPU_ATTRIBUTES

#ifndef PER_CPU_BASE_SECTION
#ifdef CONFIG_SMP
#define PER_CPU_BASE_SECTION ".data..percpu"
```

percpu section .data..percpu..page_aligned section percpu vmlinux -S .data.percpu..page_aligned section .data..percpu section percpu section .data..percpu section

```
include/asm-generic/vmlinux.lds.h

/**
 * PERCPU_INPUT - the percpu input sections
 * @cacheline: cacheline size
 *
 * The core percpu section names and core symbols which do not rely
 * directly upon load addresses.
 *
 * @cacheline is used to align subsections to avoid false cacheline
 * sharing between subsections for different purposes.
 */
#define PERCPU_INPUT(cacheline)      \
 __per_cpu_start = .;      \
 . = ALIGN(PAGE_SIZE);      \
 *(.data..percpu..page_aligned)     \
 . = ALIGN(cacheline);      \
 __per_cpu_hot_start = .;     \
 *(SORT_BY_ALIGNMENT(.data..percpu..hot.*))   \
 __per_cpu_hot_end = .;      \
 . = ALIGN(cacheline);      \
 *(.data..percpu..read_mostly)     \
 . = ALIGN(cacheline);      \
 *(.data..percpu)      \
 *(.data..percpu..shared_aligned)    \
 PERCPU_DECRYPTED_SECTION     \
 __per_cpu_end = .;

/**
 * PERCPU_SECTION - define output section for percpu area
 * @cacheline: cacheline size
 *
 * Macro which expands to output section for percpu area.
 *
 * @cacheline is used to align subsections to avoid false cacheline
 * sharing between subsections for different purposes.
 */
#define PERCPU_SECTION(cacheline)      \
 . = ALIGN(PAGE_SIZE);       \
 .data..percpu : AT(ADDR(.data..percpu) - LOAD_OFFSET) { \
  PERCPU_INPUT(cacheline)      \
 }
```

percpu section .data..percpu PERCPU_SECTION->PERCPU_INPUT section ___per_c
section

arch/x86/kernel/vmlinux.lds.S

```
PERCPU_SECTION(L1_CACHE_BYTES)
```

.data..percpu section Type PROGBITS Flags WA vmlinux bootloader percpu
offset ___per_cpu_start percpu percpu percpu
cea setup_cpu_entry_area->get_cpu_entry_area

arch/x86/mm/cpu_entry_area.c

```
/* Is called from entry code, so must be noinstr */
noinstr struct cpu_entry_area *get_cpu_entry_area(int cpu)
{
 unsigned long va = CPU_ENTRY_AREA_PER_CPU + cea_offset(cpu) * CPU_ENTRY_AREA_SIZE;
 BUILD_BUG_ON(sizeof(struct cpu_entry_area) % PAGE_SIZE != 0);

 return (struct cpu_entry_area *) va;
}
```

arch/x86/include/asm/pgtable_areas.h

```
/* Single page reserved for the readonly IDT mapping: */
#define CPU_ENTRY_AREA_RO_IDT  CPU_ENTRY_AREA_BASE
#define CPU_ENTRY_AREA_PER_CPU  (CPU_ENTRY_AREA_RO_IDT + PAGE_SIZE)
```

CPU_ENTRY_AREA_BASE CPU_ENTRY_AREA_BASE entry
area per cpu CPU_ENTRY_AREA_PER_CPU cea_set_pte per
cpu percpu cpu cea_offset
cea_set_pte per cpu gdt_page cea->gdt PAGE_KERNEL_RO x86-
64 percpu gdt page percpu back store page direct rw cea-
>gdt fixmap ro gdt page

arch/x86/include/asm/desc.h

```
/* Provide the original GDT */
static inline struct desc_struct *get_cpu_gdt_rw(unsigned int cpu)
{
 return per_cpu(gdt_page, cpu).gdt;
}
```

gdt page

```
arch/x86/include/asm/desc.h

/* Provide the fixmap address of the remapped GDT */
static inline struct desc_struct *get_cpu_gdt_ro(int cpu)
{
 return (struct desc_struct *)&get_cpu_entry_area(cpu)->gdt;
}
```

gdt page gdtr

```
arch/x86/kernel/cpu/common.c

/* Load the original GDT from the per-cpu structure */
void load_direct_gdt(int cpu)
{
 struct desc_ptr gdt_descr;

 gdt_descr.address = (long)get_cpu_gdt_rw(cpu);
 gdt_descr.size = GDT_SIZE - 1;
 load_gdt(&gdt_descr);
}
EXPORT_SYMBOL_FOR_KVM(load_direct_gdt);

/* Load a fixmap remapping of the per-cpu GDT */
void load_fixmap_gdt(int cpu)
{
 struct desc_ptr gdt_descr;

 gdt_descr.address = (long)get_cpu_gdt_ro(cpu);
 gdt_descr.size = GDT_SIZE - 1;
 load_gdt(&gdt_descr);
}
EXPORT_SYMBOL_GPL(load_fixmap_gdt);
```

gdtr load_gdt lgdt gdtr

```
arch/x86/include/asm/desc.h

#define load_gdt(dtr)    native_load_gdt(dtr)

static inline void native_load_gdt(const struct desc_ptr *dtr)
{
 asm volatile("lgdt %0"::"m" (*dtr));
}
```

Linux gdt

Linux tss Linux cpu_init_exception_handling set_tss_desc tss load_TR_desc tss register TR

arch/x86/kernel/cpu/common.c

```
/*
 * Setup everything needed to handle exceptions from the IDT, including the IST
 * exceptions which use paranoid_entry().
 */
void cpu_init_exception_handling(bool boot_cpu)
{
      ...
 set_tss_desc(cpu, &get_cpu_entry_area(cpu)->tss.x86_tss);

 load_TR_desc();
      ...
}
```

set_tss_desc tss.x86_tss gdt_page cpu_entry_area percpu backing store setup_cpu_entry_area

```
/* Setup the fixmap mappings only once per-processor */
static void __init setup_cpu_entry_area(unsigned int cpu)
{
 struct cpu_entry_area *cea = get_cpu_entry_area(cpu);
      pgprot_t tss_prot = PAGE_KERNEL_RO;

      ...
      cea_map_percpu_pages(&cea->tss, &per_cpu(cpu_tss_rw, cpu),
                      sizeof(struct tss_struct) / PAGE_SIZE, tss_prot);
      ...
}
```

gdt tss Linux tss rsp0