Most of the text from Keith Owens, hacked by AK

x86_64 page size (PAGE_SIZE) is 4K.

Like all other architectures, x86_64 has a kernel stack for every
active thread.   These thread stacks are THREAD_SIZE (2*PAGE_SIZE) big.
These stacks contain useful data as long as a thread is alive or a
zombie. While the thread is in user space the kernel stack is empty
except for the thread_info structure at the bottom.

In addition to the per thread stacks, there are specialized stacks
associated with each CPU.   These stacks are only used while the kernel
is in control on that CPU; when a CPU returns to user space the
specialized stacks contain no useful data.   The main CPU stacks are:

* Interrupt stack.   IRQSTACKSIZE

  Used for external hardware interrupts.   If this is the first external
  hardware interrupt (i.e. not a nested hardware interrupt) then the
  kernel switches from the current task to the interrupt stack.   Like
  the split thread and interrupt stacks on i386 (with CONFIG_4KSTACKS),
  this gives more room for kernel interrupt processing without having
  to increase the size of every per thread stack.

  The interrupt stack is also used when processing a softirq.

Switching to the kernel interrupt stack is done by software based on a
per CPU interrupt nest counter. This is needed because x86-64 "IST"
hardware stacks cannot nest without races.

x86_64 also has a feature which is not available on i386, the ability
to automatically switch to a new stack for designated events such as
double fault or NMI, which makes it easier to handle these unusual
events on x86_64.   This feature is called the Interrupt Stack Table
(IST).   There can be up to 7 IST entries per CPU. The IST code is an
index into the Task State Segment (TSS). The IST entries in the TSS
point to dedicated stacks; each stack can be a different size.

An IST is selected by a non-zero value in the IST field of an
interrupt-gate descriptor.   When an interrupt occurs and the hardware
loads such a descriptor, the hardware automatically sets the new stack
pointer based on the IST value, then invokes the interrupt handler.   If
software wants to allow nested IST interrupts then the handler must
adjust the IST values on entry to and exit from the interrupt handler.
(This is occasionally done, e.g. for debug exceptions.)

Events with different IST codes (i.e. with different stacks) can be
nested.   For example, a debug interrupt can safely be interrupted by an
NMI.   arch/x86_64/kernel/entry.S::paranoidentry adjusts the stack
pointers on entry to and exit from all IST events, in theory allowing
IST events with the same code to be nested.   However in most cases, the
stack size allocated to an IST assumes no nesting for the same code.
If that assumption is ever broken then the stacks will become corrupt.

The currently assigned IST stacks are :-

* STACKFAULT_STACK.   EXCEPTION_STKSZ (PAGE_SIZE).

  Used for interrupt 12 - Stack Fault Exception (#SS).

  This allows the CPU to recover from invalid stack segments. Rarely
  happens.

* DOUBLEFAULT_STACK.   EXCEPTION_STKSZ (PAGE_SIZE).

  Used for interrupt 8 - Double Fault Exception (#DF).

  Invoked when handling one exception causes another exception. Happens
  when the kernel is very confused (e.g. kernel stack pointer corrupt).
  Using a separate stack allows the kernel to recover from it well enough
  in many cases to still output an oops.

* NMI_STACK.   EXCEPTION_STKSZ (PAGE_SIZE).

  Used for non-maskable interrupts (NMI).

  NMI can be delivered at any time, including when the kernel is in the
  middle of switching stacks. Using IST for NMI events avoids making
  assumptions about the previous state of the kernel stack.

* DEBUG_STACK.   DEBUG_STKSZ

  Used for hardware debug interrupts (interrupt 1) and for software
  debug interrupts (INT3).

  When debugging a kernel, debug interrupts (both hardware and
  software) can occur at any time. Using IST for these interrupts
  avoids making assumptions about the previous state of the kernel
  stack.

* MCE_STACK.   EXCEPTION_STKSZ (PAGE_SIZE).

  Used for interrupt 18 - Machine Check Exception (#MC).

  MCE can be delivered at any time, including when the kernel is in the
  middle of switching stacks. Using IST for MCE events avoids making
  assumptions about the previous state of the kernel stack.

For more details see the Intel IA32 or AMD AMD64 architecture manuals.