LECCIÓN 2: INTERRUPCIONES SOFTWARE

LECCIÓN 2: INTERRUPCIONES SOFTWARE

2.1 Introducción

Una interrupción se genera cuando se quiere que la CPU deje de ejecutar el proceso en curso y ejecute una función específica de quien produce la interrupción. Cuando se ejecuta esta función específica decimos que la CPU está atendiendo la interrupción. Podemos realizar una clasificación de las interrupciones, atendiendo a la fuente que las produce.

Interrupcion software, se produce cuando un usuario solicita una llamada del sistema.

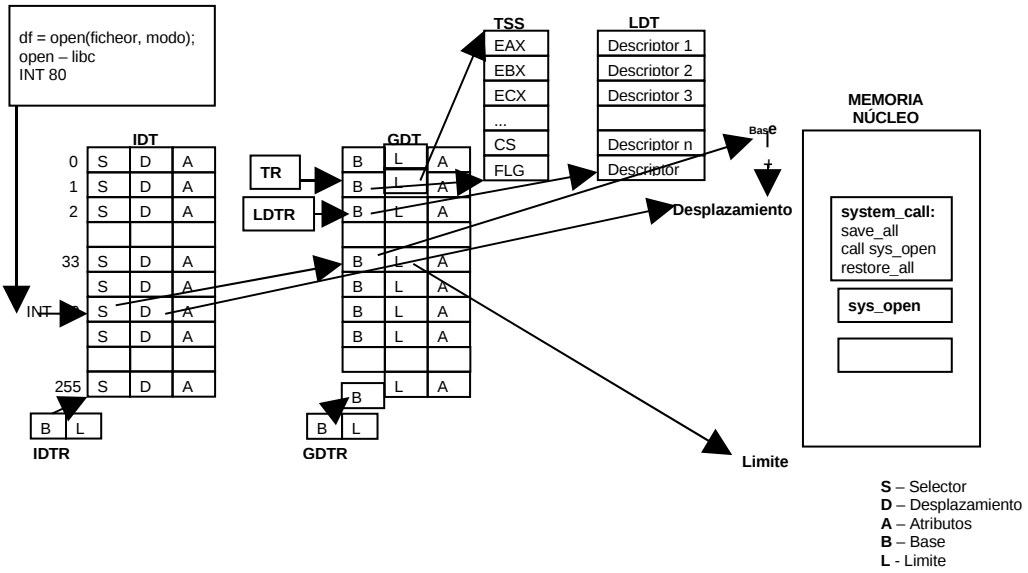
Interrupciones hardware, son causadas cuando un dispositivo hardware requiere la atención de la CPU para que se ejecute su manejador.

Excepciones, son interrupciones causadas por la propia CPU, cuando ocurre algún suceso, por ejemplo una división por cero.

Una Interrupción software se produce cuando un usuario solicita un recurso del núcleo, mediante una llamada al sistema, open, write, read, mount, etc. Los pasos que se producen son los siguientes:
El proceso usuario solicita la función correspondiente de la librería libc, que ha sido añadida en la compilación del proceso.
df = open (fichero, modo);
La función de librería coloca los parámetros de la llamada en los registros del procesador y ejecuta la instrucción **INT 0x80.**
Se conmuta de modo usuario a modo núcleo mediante las estructuras conocidas como las tablas IDT (Tabla Descriptora de Interrupciones) y GDT (Tabla Global de Descriptores).
Entra a ejecutarse una función del núcleo, **system_call**, Interfase entre el usuario y el núcleo.
Cuando se termina la llamada, system_call retorna al proceso que la llamo y se retorna a modo usuario.

El modo de llegar a la función system_call se visualiza en la siguiente figura.

Llega la interrupción software INT 80h y se busca en la IDT la entrada correspondiente a la interrupción.
Se obtiene un puntero desplazamiento al núcleo (D).
Y una entrada (S) a GDT.
La entrada en la GDT indica una base (B) y un limite (L) del núcleo.
Se llega al manejador de interrupción system_call.



IDT (Tabla Descriptora de Interrupciones)
Tabla que guarda los descriptores de interrupciones, cada vez que se produce una interrupción se salta a una entrada de esta tabla. Contiene 256 entradas, y cada entrada tiene información para llegar a una función manejadora de la interrupción. Contienen el campo denominado Selector, que tiene la dirección de una entrada de la tabla GDT; el campo Desplazamiento, que contiene el desplazamiento que hay que sumar a la base del núcleo para llegar a la función que queremos que se ejecute cuando se interrumpe; por último contiene una serie de campos denominados Atributos, que fijan si el desplazamiento viene en bytes o en palabras, bits de protección, etc). Existe un registro IDTR dentro del microprocesador que contiene la dirección base y la longitud de esta tabla, para encontrarla con rapidez. Existe una tabla para todo el sistema. Para rellenar esta tabla se precisan instrucciones en ensamblador privilegiadas, un usuario normal no puede modificar sus entradas.

GDT (Tabla Global de Descriptores)
Esta tabla es única para el sistema y contiene información de los descriptores de segmentos del sistema, los microprocesadores de Intel obligan a trabajar con segmentos de memoria (base, longitud). Una de las entradas contiene la base y la longitud del segmento que contiene el núcleo del sistema operativo en memoria principal. Cada entrada también tiene otros campos que denominamos Atributos, que definen atributos del segmentos, como permisos de acceso a ese segmento. Dentro de la CPU existe un registro especifico GDTR para almacenar la dirección base y longitud de esta tabla y facilitar su acceso. Otras entradas de esta tabla describen las tablas LDT y el segmento TSS.

LDT (Tabla Local de Descriptores)
Existe una tabla por cada proceso que hay en el sistema, si bien solo una esta activa, la del proceso que se está ejecutando en la CPU. Contiene distintos descriptores de

segmentos correspondientes al proceso, como el segmento de código, el segmento de datos, el segmento stack, etc.

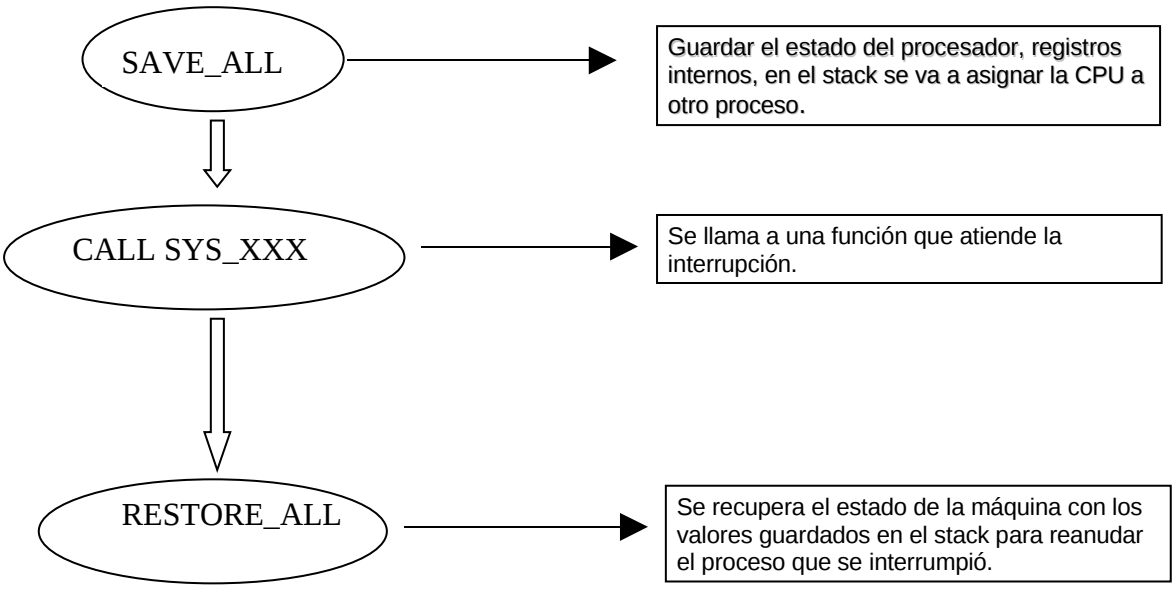
TSS
Esta tabla tiene una imagen de todos los registros de la CPU relacionados con un proceso, (EAX, EBX, ECX, …), cada vez que se produce una interrupción, la CPU de forma automática, copia el estado de sus registros en este segmento.

2.2 system_call

Estudiemos esta función en el núcleo version 2.6.28.6, para la plataforma Intel de 32 bits.

La encontramos en la ruta arch/x86/kernel/entry_32_S.

Como todas las interrupciones tiene tres puntos principales que en este caso los realiza de la siguiente forma



ENTRY(system_call)

/* el primer argumento de **system_call** es el número de la llamada al sistema
 * a invocar; se ha colocado por la función de la libreria en el registro EAX.
 * System_call permite hasta cuatro argumentos mas que serán pasados a
 * lo largo de la llamada al sistema. Incluso uno de los argumentos puede ser
 * un puntero a una estructura */

```
ENTRY(system_call)
        pushl %eax                      # save orig_eax
```

/* SAVE_ALL hace una copia de todos los registros de la CPU en la pila.
* Cualquier función de C que sea llamada encontrará sus argumentos en
* la pila porque SAVE_ALL los pone allí */

```
        SAVE_ALL

        GET_THREAD_INFO(%ebp)
```

/* Comprueba si la llamada al sistema esta haciendo una traza. Algunos
 * programas permiten hacer trazas en las llamadas al sistema como una
 * herramienta para curiosos o para obtener información extra de depuración */

```
/* Note, _TIF_SECCOMP is bit number 8, and so it needs testw and not testb
*/
        testw $(_TIF_SYSCALL_EMU|_TIF_SYSCALL_TRACE|_TIF_SECCOMP|_TIF_SYSCALL_AUDIT),
TI_flags(%ebp)
        jnz syscall_trace_entry
```

/* Compara si el numero de la llamada al sistema (EAX) excede del numero
 * máximo de llamadas al sistema */

```
        cmpl $(nr_syscalls), %eax
        jae syscall_badsys
```

/* Llama a la función del sistema **sys_call_table** que es una tabla de punteros
 * a funciones del núcleo que implementan las llamadas al sistema. */

/* El segundo conjunto de paréntesis en la línea contiene tres argumentos:
 * la dirección base del vector,
 * el índice (EAX, el numero de la llamada al sistema),
 * y la escala o numero de bytes en cada elemento del vector. */

```
syscall_call:
        call *sys_call_table(,%eax,4)
```

/* La llamada al sistema ha retornado. El valor devuelto que está en el
 * registro eax, se guarda en la posición EAX de la pila de modo
 * que RESTORE_ALL pueda restaurar el verdadero registro eax de
 * la pila, además del resto de los registros. */

```
        movl %eax,PT_EAX(%esp)             # store the return value


syscall_exit:
```

/* Deshabilita las interrupciones */

```
LOCKDEP_SYS_EXIT
DISABLE_INTERRUPTS(CLBR_ANY)# make sure we don't miss an interrupt
                                # setting need_resched or sigpending
                                # between sampling and the iret
        TRACE_IRQS_OFF
```

/* Analiza si es necesario volver asignar la CPU
 * o si existen señales pendientes */

```
        movl TI_flags(%ebp), %ecx
        testw $_TIF_ALLWORK_MASK, %cx   # current->work
        jne syscall_exit_work
```

/* La etiqueta **restore_all** es el punto de salida para **system_call**.
 * Su contenido es simple como la macro **RESTORE_REGS**, que
 * recupera los argumentos guardados por **SAVE_ALL** y retorna
 * al llamador del **system_call**. */

```
restore_all:
        movl PT_EFLAGS(%esp), %eax      # mix EFLAGS, SS and CS
        # Warning: PT_OLDSS(%esp) contains the wrong/random values if we
        # are returning to the kernel.
        # See comments in process.c:copy_thread() for details.
        movb PT_OLDSS(%esp), %ah
        movb PT_CS(%esp), %al
        andl $(VM_MASK | (SEGMENT_TI_MASK << 8) | SEGMENT_RPL_MASK), %eax
        cmpl $((SEGMENT_LDT << 8) | USER_RPL), %eax
        CFI_REMEMBER_STATE
        je ldt_ss                       # returning to user-space with LDT SS
restore_nocheck:
        TRACE_IRQS_IRET
restore_nocheck_notrace:

        RESTORE_REGS  # macro para recuperar los registros

 addl $4, %esp                  # skip orig_eax/error_code
        CFI_ADJUST_CFA_OFFSET -4
 irq_return:
        INTERRUPT_RETUR
.
.
Exite más código hasta llegar a

ENDPROC(system_call)
```

/* Puntos de entrada desde otros lugares */

/* Comprueba si hay trabajo pendiente antes de retornar */,

```
         # perform work that needs to be done immediately before resumption
         ALIGN
work_pending:
         testb $_TIF_NEED_RESCHED, %cl
         jz work_notifysig
```

/* se necesita llamar al asignador de la cpu, si ha interrumpido un proceso
 * de mayor prioridad que el actual o se ha terminado la rodaja de tiempo */

```
work_resched:
call schedule
     LOCKDEP_SYS_EXIT
     DISABLE_INTERRUPTS(CLBR_ANY)    # make sure we don't miss an interrupt
                                     # setting need_resched or sigpending
                                     # between sampling and the iret
      TRACE_IRQS_OFF
```

/* Comprueba si no existe trabajo pendiente */

```
        movl TI_flags(%ebp), %ecx
        andl $_TIF_WORK_MASK, %ecx     # is there any work to be done other
                                       # than syscall tracing?
        jz restore_all
```

/* Comprueba si se tiene que volver a asignar la CPU */

```
        testb $_TIF_NEED_RESCHED, %cl
        jnz work_resched
```

/* Comprueba si tiene señales pendientes y vuelve al modo núcleo modo virtual */

```
work_notifysig:                     # deal with pending signals and
                                    # notify-resume requests
#ifdef CONFIG_VM86
         testl $X86_EFLAGS_VM, PT_EFLAGS(%esp)
         movl %esp, %eax
         jne work_notifysig_v86       # returning to kernel-space or
                                      # vm86-space
```

/* Comprueba si existe trabajo pendiente */

```
         xorl %edx, %edx
         call do_notify_resume
         jmp resume_userspace_sig

         ALIGN
```

/* Si retornamos a modo virtual llamamos a save_v86_state */

```
work_notifysig_v86:
         pushl %ecx                 # save ti_flags for do_notify_resume
         call save_v86_state        # %eax contains pt_regs pointer
         popl %ecx
         CFI_ADJUST_CFA_OFFSET -4
         movl %eax, %esp
#else
         movl %esp, %eax
#endif
         xorl %edx, %edx
         call do_notify_resume
         jmp resume_userspace_sig
END(work_pending)
```

/* Punto de entrada si la llamada al sistema del proceso actual
 * esta haciendo una traza */

```
         # perform syscall exit tracing
         ALIGN
syscall_trace_entry:
         movl $-ENOSYS,EAX(%esp)
         movl %esp, %eax
         call syscall_trace_enter
/* What it returned is what we'll actually use.  */
         cmpl $(nr_syscalls), %eax
         jnae syscall_call
         jmp syscall_exit
```

END(syscall_trace_entry)

```
         # perform syscall exit tracing
         ALIGN
```

/* comprueba si hay trabajo pendiente si se vuelve del modo traza */

```
syscall_exit_work:
         testb $(_TIF_SYSCALL_TRACE|_TIF_SYSCALL_AUDIT|_TIF_SINGLESTEP), %cl
         jz work_pending
```

/* Habilita las interrupciones */

```
         ENABLE_INTERRUPTS(CLBR_ANY)  # could let do_syscall_trace() call
                                      # schedule() instead
         movl %esp, %eax
```

/* se llama al modo traza */

```
        call do_syscall_trace_leave
```
/* se vuelve al modo usuario */
```
        jmp resume_userspace
END(syscall_exit_work)
        CFI_ENDPROC

        RING0_INT_FRAME           # can't unwind into user space anyway
```

/* si se produce un error guardar el estado, obtener la información
 * del hilo y volver al espacio de usuario */
```
syscall_fault:
        GET_THREAD_INFO(%ebp)
        movl $-EFAULT,EAX(%esp)
        jmp resume_userspace
END(syscall_fault)
```

/* si la llamada no es correcta volver al espacio de usuario */
```
syscall_badsys:
        movl $-ENOSYS,EAX(%esp)
        jmp resume_userspace
END(syscall_badsys)
        CFI_ENDPROC

#define FIXUP_ESPFIX_STACK \
 599        /* since we are on a wrong stack, we cant make it a C code :
( */ \
 600        PER_CPU(gdt_page, %ebx); \
 601        GET_DESC_BASE(GDT_ENTRY_ESPFIX_SS, %ebx, %eax, %ax, %al,
%ah); \
 602        addl %esp, %eax; \
 603        pushl $__KERNEL_DS; \
 604        CFI_ADJUST_CFA_OFFSET 4; \
 605        pushl %eax; \
 606        CFI_ADJUST_CFA_OFFSET 4; \
 607        lss (%esp), %esp; \
 608        CFI_ADJUST_CFA_OFFSET -8;
 609#define UNWIND_ESPFIX_STACK \
 610        movl %ss, %eax; \
 611        /* see if on espfix stack */ \
 612        cmpw $__ESPFIX_SS, %ax; \
 613        jne 27f; \
 614        movl $__KERNEL_DS, %eax; \
 615        movl %eax, %ds; \
 616        movl %eax, %es; \
 617        /* switch to normal stack */ \
 618        FIXUP_ESPFIX_STACK; \
 61927:;
```

Posiciones de los registros en el stack , después de volver de una llamada, muchos procedimientos necesitan esta información en el stack.

El registro EBX contiene la variable “current” la dirección que apunta a la entrada en la tabla de procesos del proceso que realiza la llamada.

EBX ECX EDX ESI EDI EBP EAX DS ES ORIG_EAX EIP CS EFLAGS OLDESP OLDSS	= 0x00 = 0x04 = 0x08 = 0x0C = 0x10 = 0x14 = 0x18 = 0x1C = 0x20 = 0x24 = 0x28 = 0x2C = 0x30 = 0x34 = 0x38	CF_MASK IF_MASK NT_MASK VM_MASK	= 0x00000001 = 0x00000200 = 0x00004000 = 0x00020000
		/* * these are offsets into the task-struct. */	
		state flags sigpending addr_limit exec_domain need_resched	= 0 = 4 = 8 = 12 = 16 = 20

```
 104#define SAVE_ALL \
```

SAVE_ALL
/* Guarda los registros en el stack */

```
#define SAVE_ALL
   cld;
   pushl %es;
   pushl %ds;
   pushl %eax;
   pushl %ebp;
   pushl %edi;
   pushl %esi;
   pushl %edx;
   pushl %ecx;
   pushl %ebx;
   movl $(__USER_DS),%edx;
   movl %dx,%ds;
   movl %dx,%es;
   movl $(__KERNEL_PERCPU), %edx; \
   movl %edx, %fs
```

RESTORE
/* Recupera registros desde el stack */

```
 142#define RESTORE_INT_REGS \
         popl %ebx;      \
         popl %ecx;      \
         popl %edx;      \
         popl %esi;      \
         popl %edi;      \
         popl %ebp;      \
         popl %eax
```

```
#define RESTORE_REGS; \
        RESTORE_INT_REGS \
1:      popl %ds;       \
2:      popl %es;
3:      popl %fs;       \
.section .fixup,"ax";   \
4:      movl $0,(%esp); \
        jmp 1b;         \
5:      movl $0,(%esp); \
        jmp 2b;         \
6:      movl $0,(%esp); \
        jmp 3b;         \
.section __ex_table,"a";\
         .align 4;       \
         .long 1b,4b;    \
         .long 2b,5b;    \
         .long 3b,6b;    \
.popsection
```

2.3 Sys_call_table

Tabla con todas la funciones que tratan las llamadas al sistema, se encuentra en el mismo directorio en el fichero syscall_table_32.S

```
ENTRY(sys_call_table)
        .long sys_restart_syscall  /* 0 - old "setup()" system call, used
for restarting */
        .long sys_exit
        .long sys_fork
        .long sys_read
        .long sys_write
        .long sys_open          /* 5 */
        .long sys_close
        .long sys_waitpid
        .long sys_creat
        .long sys_link
        .long sys_unlink        /* 10 */
        .long sys_execve
        .long sys_chdir
        .long sys_time
        .long sys_mknod
        .long sys_chmod         /* 15 */
        .long sys_lchown16
        .long sys_ni_syscall    /* old break syscall holder */
        .long sys_stat
        .long sys_lseek
        .long sys_getpid        /* 20 */
        .long sys_mount
        .long sys_oldumount
        .long sys_setuid16
        .long sys_getuid16
        .long sys_stime         /* 25 */
        .long sys_ptrace
        .long sys_alarm
        .long sys_fstat
        .long sys_pause
```

```
        .long sys_utime         /* 30 */
        .long sys_ni_syscall    /* old stty syscall holder */
        .long sys_ni_syscall    /* old gtty syscall holder */
        .long sys_access
        .long sys_nice
        .long sys_ni_syscall    /* 35 - old ftime syscall holder */
        .long sys_sync
        .long sys_kill
        .long sys_rename
        .long sys_mkdir
        .long sys_rmdir         /* 40 */
        .long sys_dup
        .long sys_pipe
        .long sys_times
        .long sys_ni_syscall    /* old prof syscall holder */
        .long sys_brk           /* 45 */
        .long sys_setgid16
        .long sys_getgid16
        .long sys_signal
        .long sys_geteuid16
        .long sys_getegid16     /* 50 */
        .long sys_acct
        .long sys_umount        /* recycled never used phys() */
        .long sys_ni_syscall    /* old lock syscall holder */
        .long sys_ioctl
        .long sys_fcntl         /* 55 */
        .long sys_ni_syscall    /* old mpx syscall holder */
        .long sys_setpgid
        .long sys_ni_syscall    /* old ulimit syscall holder */
        .long sys_olduname
        .long sys_umask         /* 60 */
        .long sys_chroot
        .long sys_ustat
        .long sys_dup2
        .long sys_getppid
        .long sys_getpgrp       /* 65 */
        .long sys_setsid
        .long sys_sigaction
        .long sys_sgetmask
        .long sys_ssetmask
        .long sys_setreuid16    /* 70 */
        .long sys_setregid16
        .long sys_sigsuspend
        .long sys_sigpending
        .long sys_sethostname
        .long sys_setrlimit     /* 75 */
        .long sys_old_getrlimit
        .long sys_getrusage
        .long sys_gettimeofday
        .long sys_settimeofday
        .long sys_getgroups16   /* 80 */
        .long sys_setgroups16
        .long old_select
        .long sys_symlink
        .long sys_lstat
        .long sys_readlink      /* 85 */
        .long sys_uselib
        .long sys_swapon
        .long sys_reboot
        .long sys_old_readdir
        .long old_mmap          /* 90 */
```

```
        .long sys_munmap
        .long sys_truncate
        .long sys_ftruncate
        .long sys_fchmod
        .long sys_fchown16      /* 95 */
        .long sys_getpriority
        .long sys_setpriority
        .long sys_ni_syscall    /* old profil syscall holder */
        .long sys_statfs
        .long sys_fstatfs       /* 100 */
        .long sys_ioperm
        .long sys_socketcall
        .long sys_syslog
        .long sys_setitimer
        .long sys_getitimer     /* 105 */
        .long sys_newstat
        .long sys_newlstat
        .long sys_newfstat
        .long sys_uname
        .long sys_iopl          /* 110 */
        .long sys_vhangup
        .long sys_ni_syscall    /* old "idle" system call */
        .long sys_vm86old
        .long sys_wait4
        .long sys_swapoff       /* 115 */
        .long sys_sysinfo
        .long sys_ipc
        .long sys_fsync
        .long sys_sigreturn
        .long sys_clone         /* 120 */
        .long sys_setdomainname
        .long sys_newuname
        .long sys_modify_ldt
        .long sys_adjtimex
        .long sys_mprotect      /* 125 */
        .long sys_sigprocmask
        .long sys_ni_syscall    /* old "create_module" */
        .long sys_init_module
        .long sys_delete_module
        .long sys_ni_syscall    /* 130: old "get_kernel_syms" */
        .long sys_quotactl
        .long sys_getpgid
        .long sys_fchdir
        .long sys_bdflush
        .long sys_sysfs         /* 135 */
        .long sys_personality
        .long sys_ni_syscall    /* reserved for afs_syscall */
        .long sys_setfsuid16
        .long sys_setfsgid16
        .long sys_llseek        /* 140 */
        .long sys_getdents
        .long sys_select
        .long sys_flock
        .long sys_msync
        .long sys_readv         /* 145 */
        .long sys_writev
        .long sys_getsid
        .long sys_fdatasync
        .long sys_sysctl
        .long sys_mlock         /* 150 */
        .long sys_munlock
```

```
        .long sys_mlockall
        .long sys_munlockall
        .long sys_sched_setparam
        .long sys_sched_getparam   /* 155 */
        .long sys_sched_setscheduler
        .long sys_sched_getscheduler
        .long sys_sched_yield
        .long sys_sched_get_priority_max
        .long sys_sched_get_priority_min  /* 160 */
        .long sys_sched_rr_get_interval
        .long sys_nanosleep
        .long sys_mremap
        .long sys_setresuid16
        .long sys_getresuid16   /* 165 */
        .long sys_vm86
        .long sys_ni_syscall    /* Old sys_query_module */
        .long sys_poll
        .long sys_nfsservctl
        .long sys_setresgid16   /* 170 */
        .long sys_getresgid16
        .long sys_prctl
        .long sys_rt_sigreturn
        .long sys_rt_sigaction
        .long sys_rt_sigprocmask        /* 175 */
        .long sys_rt_sigpending
        .long sys_rt_sigtimedwait
        .long sys_rt_sigqueueinfo
        .long sys_rt_sigsuspend
        .long sys_pread64       /* 180 */
        .long sys_pwrite64
        .long sys_chown16
        .long sys_getcwd
        .long sys_capget
        .long sys_capset        /* 185 */
        .long sys_sigaltstack
        .long sys_sendfile
        .long sys_ni_syscall    /* reserved for streams1 */
        .long sys_ni_syscall    /* reserved for streams2 */
        .long sys_vfork         /* 190 */
        .long sys_getrlimit
        .long sys_mmap2
        .long sys_truncate64
        .long sys_ftruncate64
        .long sys_stat64        /* 195 */
        .long sys_lstat64
        .long sys_fstat64
        .long sys_lchown
        .long sys_getuid
        .long sys_getgid        /* 200 */
        .long sys_geteuid
        .long sys_getegid
        .long sys_setreuid
        .long sys_setregid
        .long sys_getgroups     /* 205 */
        .long sys_setgroups
        .long sys_fchown
        .long sys_setresuid
        .long sys_getresuid
        .long sys_setresgid     /* 210 */
        .long sys_getresgid
        .long sys_chown
```

```
        .long sys_setuid
        .long sys_setgid
        .long sys_setfsuid      /* 215 */
        .long sys_setfsgid
        .long sys_pivot_root
        .long sys_mincore
        .long sys_madvise
        .long sys_getdents64    /* 220 */
        .long sys_fcntl64
        .long sys_ni_syscall    /* reserved for TUX */
        .long sys_ni_syscall
        .long sys_gettid
        .long sys_readahead     /* 225 */
        .long sys_setxattr
        .long sys_lsetxattr
        .long sys_fsetxattr
       .long sys_getxattr
        .long sys_lgetxattr     /* 230 */
        .long sys_fgetxattr
        .long sys_listxattr
        .long sys_llistxattr
        .long sys_flistxattr
        .long sys_removexattr   /* 235 */
        .long sys_lremovexattr
        .long sys_fremovexattr
        .long sys_tkill
        .long sys_sendfile64
        .long sys_futex         /* 240 */
        .long sys_sched_setaffinity
        .long sys_sched_getaffinity
        .long sys_set_thread_area
        .long sys_get_thread_area
        .long sys_io_setup      /* 245 */
        .long sys_io_destroy
        .long sys_io_getevents
        .long sys_io_submit
        .long sys_io_cancel
        .long sys_fadvise64     /* 250 */
        .long sys_ni_syscall
        .long sys_exit_group
        .long sys_lookup_dcookie
        .long sys_epoll_create
        .long sys_epoll_ctl     /* 255 */
        .long sys_epoll_wait
        .long sys_remap_file_pages
        .long sys_set_tid_address
        .long sys_timer_create
        .long sys_timer_settime         /* 260 */
        .long sys_timer_gettime
        .long sys_timer_getoverrun
        .long sys_timer_delete
        .long sys_clock_settime
        .long sys_clock_gettime         /* 265 */
        .long sys_clock_getres
        .long sys_clock_nanosleep
        .long sys_statfs64
        .long sys_fstatfs64
        .long sys_tgkill        /* 270 */
        .long sys_utimes
        .long sys_fadvise64_64
        .long sys_ni_syscall    /* sys_vserver */
```

```
        .long sys_mbind
        .long sys_get_mempolicy
        .long sys_set_mempolicy
        .long sys_mq_open
        .long sys_mq_unlink
        .long sys_mq_timedsend
        .long sys_mq_timedreceive       /* 280 */
        .long sys_mq_notify
        .long sys_mq_getsetattr
        .long sys_kexec_load
        .long sys_waitid
        .long sys_ni_syscall            /* 285 */ /* available */
        .long sys_add_key
        .long sys_request_key
        .long sys_keyctl
        .long sys_ioprio_set
        .long sys_ioprio_get            /* 290 */
        .long sys_inotify_init
        .long sys_inotify_add_watch
        .long sys_inotify_rm_watch
        .long sys_migrate_pages
        .long sys_openat                /* 295 */
        .long sys_mkdirat
        .long sys_mknodat
        .long sys_fchownat
        .long sys_futimesat
        .long sys_fstatat64             /* 300 */
        .long sys_unlinkat
        .long sys_renameat
        .long sys_linkat
        .long sys_symlinkat
        .long sys_readlinkat            /* 305 */
        .long sys_fchmodat
        .long sys_faccessat
        .long sys_pselect6
        .long sys_ppoll
        .long sys_unshare               /* 310 */
        .long sys_set_robust_list
        .long sys_get_robust_list
        .long sys_splice
        .long sys_sync_file_range
        .long sys_tee                   /* 315 */
        .long sys_vmsplice
        .long sys_move_pages
        .long sys_getcpu
        .long sys_epoll_pwait
        .long sys_utimensat             /* 320 */
        .long sys_signalfd
        .long sys_timerfd_create
        .long sys_eventfd
        .long sys_fallocate
        .long sys_timerfd_settime       /* 325 */
        .long sys_timerfd_gettime
        .long sys_signalfd4
        .long sys_eventfd2
        .long sys_epoll_create1
        .long sys_dup3                  /* 330 */
        .long sys_pipe2
        .long sys_inotify_init1
```

2.4 Excepciones

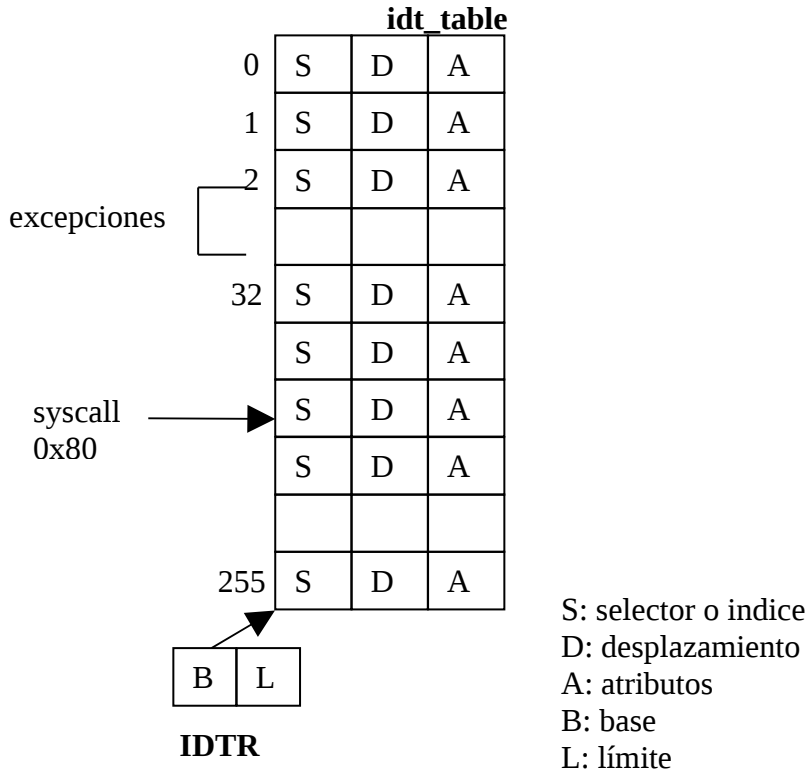
Son interrupciones producidas por la propia CPU cuando se producen ciertas situaciones como división por cero, desbordamiento del stack, fallo de página etc.

Como todas las interrupciones el núcleo realiza los siguientes pasos:

1. Guardar el estado del proceso en el stack.
   Pushl, pushl,…
2. Llamar la función manejadora de la excepción.
   Call …
3. Recuperar el estado de la función.
   Pop, pop, …

Cuando se produce una excepción, la propia CPU realiza una instrucción de interrupción INT 0xNN para saltar dentro de la tabla IDT tabla de descriptores de segmento de interrupciones en la posición NN.

En esta posición se encuentra un selector a la GDT tabla global de descriptores y un desplazamiento para alcanzar la dirección dentro del núcleo de la función que hay que ejecutar cuando se produce esta excepción.



/* punto de entrada cuando se produce un fallo de página, esto es la CPU necesita datos y no están en memoria principal y necesita traerlos de memoria segundaria */

```
 681KPROBE_ENTRY(page_fault)
        RING0_EC_FRAME
        pushl $do_page_fault
        CFI_ADJUST_CFA_OFFSET 4
        ALIGN
error_code:
/* La dirección de la función está en el stack en la posición %fs */
           pushl %es
         pushl %ds
         pushl %eax
         pushl %ebp
         pushl %edi
         pushl %esi
         pushl %edx
         pushl %ecx
         pushl %ebx
         cld
         pushl %fs
         movl $(__KERNEL_PERCPU), %ecx
         movl %ecx, %fs
         popl %ecx
/* almacenar la dirección de la función que atiende la excepción en el registro edi */
        /*CFI_REGISTER es, ecx*/
         movl PT_FS(%esp), %edi          # get the function address
         movl PT_ORIG_EAX(%esp), %edx    # get the error code
         movl $-1, PT_ORIG_EAX(%esp)     # no syscall to restart
         mov  %ecx, PT_FS(%esp)
/* preparar registros de la CPU para realizar la llamada */
         movl $(__USER_DS), %ecx
         movl %ecx, %ds
         movl %ecx, %es
         movl %esp,%eax                  # pt_regs pointer

/* llamada a la función que atiende la excepción */
         call *%edi

/* recuperar el proceso interrumpido */
         jmp ret_from_exception
         CFI_ENDPROC
KPROBE_END(page_fault)
```

Veamos como se recupera el estado del proceso interrumpido en una excepción:

```
246 ret_from_exception:
/* desabilita interrupciones preempt_stop se define como cli */
         preempt_stop(CLBR_ANY)
ret_from_intr:
         GET_THREAD_INFO(%ebp)
check_userspace:
        movl PT_EFLAGS(%esp), %eax      # mix EFLAGS and CS
        movb PT_CS(%esp), %al
        andl $(X86_EFLAGS_VM | SEGMENT_RPL_MASK), %eax
        cmpl $USER_RPL, %eax
        jb resume_kernel                # not returning to v8086 or
userspace
```

```
ENTRY(resume_userspace)

        DISABLE_INTERRUPTS(CLBR_ANY)# make sure we don't miss an interrupt
                                    # setting need_resched or sigpending
                                    # between sampling and the iret
        TRACE_IRQS_OFF

        movl TI_flags(%ebp), %ecx
        andl $_TIF_WORK_MASK, %ecx      # is there any work to be done on
                                        # int/exception return?
        jne work_pending
```

/* salta a recuperar el estado del proceso interrumpido */

```
        jmp restore_all
END(ret_from_exception)
```

Veamos el resto de puntos de entrada en el núcleo de las excepciones.

En un gran número de ellas, vemos que se almacena en el stack el número de la interrupción, se almacena la función que se tiene que ejecutar y se produce un salto a la etiqueta jmp error_code, que está dentro de page_fault para optimizar código y no volver a escribirlo.

```
 740ENTRY(coprocessor_error)
 741        RING0_INT_FRAME
 742        pushl $0
 743        CFI_ADJUST_CFA_OFFSET 4
 744        pushl $do_coprocessor_error
 745        CFI_ADJUST_CFA_OFFSET 4
 746        jmp error_code
 747        CFI_ENDPROC
 748END(coprocessor_error)
 749
 750ENTRY(simd_coprocessor_error)
 751        RING0_INT_FRAME
 752        pushl $0
 753        CFI_ADJUST_CFA_OFFSET 4
 754        pushl $do_simd_coprocessor_error
 755        CFI_ADJUST_CFA_OFFSET 4
 756        jmp error_code
 757        CFI_ENDPROC
 758END(simd_coprocessor_error)
 759
 760ENTRY(device_not_available)
 761        RING0_INT_FRAME
 762        pushl $-1                       # mark this as an int
 763        CFI_ADJUST_CFA_OFFSET 4
 764        pushl $do_device_not_available
 765        CFI_ADJUST_CFA_OFFSET 4
 766        jmp error_code
 767        CFI_ENDPROC
 768END(device_not_available)
 769
 770/*
 771 * Debug traps and NMI can happen at the one SYSENTER instruction
 772 * that sets up the real kernel stack. Check here, since we can't
 773 * allow the wrong stack to be used.
 774 *
```

```
 775 * "TSS_sysenter_sp0+12" is because the NMI/debug handler will have
 776 * already pushed 3 words if it hits on the sysenter instruction:
 777 * eflags, cs and eip.
 778 *
 779 * We just load the right stack, and push the three (known) values
 780 * by hand onto the new stack - while updating the return eip past
 781 * the instruction that would have done it for sysenter.
 782 */
 783#define FIX_STACK(offset, ok, label)           \
 784        cmpw $__KERNEL_CS,4(%esp);             \
 785        jne ok;                                \
 786label:                                         \
 787        movl TSS_sysenter_sp0+offset(%esp),%esp;       \
 788        CFI_DEF_CFA esp, 0;                    \
 789        CFI_UNDEFINED eip;                     \
 790        pushfl;                                \
 791        CFI_ADJUST_CFA_OFFSET 4;               \
 792        pushl $__KERNEL_CS;                    \
 793        CFI_ADJUST_CFA_OFFSET 4;               \
 794        pushl $sysenter_past_esp;              \
 795        CFI_ADJUST_CFA_OFFSET 4;               \
 796        CFI_REL_OFFSET eip, 0
 797
 798KPROBE_ENTRY(debug)
 799        RING0_INT_FRAME
 800        cmpl $ia32_sysenter_target,(%esp)
 801        jne debug_stack_correct
 802        FIX_STACK(12, debug_stack_correct, debug_esp_fix_insn)
 803debug_stack_correct:
 804        pushl $-1                       # mark this as an int
 805        CFI_ADJUST_CFA_OFFSET 4
 806        SAVE_ALL
 807        TRACE_IRQS_OFF
 808        xorl %edx,%edx                  # error code 0
 809        movl %esp,%eax                  # pt_regs pointer
 810        call do_debug
 811        jmp ret_from_exception
 812        CFI_ENDPROC
 813KPROBE_END(debug)
 814
 815/*
 816 * NMI is doubly nasty. It can happen _while_ we're handling
 817 * a debug fault, and the debug fault hasn't yet been able to
 818 * clear up the stack. So we first check whether we got  an
 819 * NMI on the sysenter entry path, but after that we need to
 820 * check whether we got an NMI on the debug path where the debug
 821 * fault happened on the sysenter path.
 822 */
 823KPROBE_ENTRY(nmi)
 824        RING0_INT_FRAME
 825        pushl %eax
 826        CFI_ADJUST_CFA_OFFSET 4
 827        movl %ss, %eax
 828        cmpw $__ESPFIX_SS, %ax
 829        popl %eax
 830        CFI_ADJUST_CFA_OFFSET -4
 831        je nmi_espfix_stack
 832        cmpl $ia32_sysenter_target,(%esp)
 833        je nmi_stack_fixup
 834        pushl %eax
 835        CFI_ADJUST_CFA_OFFSET 4
```

```
 836        movl %esp,%eax
 837        /* Do not access memory above the end of our stack page,
 838         * it might not exist.
 839         */
 840        andl $(THREAD_SIZE-1),%eax
 841        cmpl $(THREAD_SIZE-20),%eax
 842        popl %eax
 843        CFI_ADJUST_CFA_OFFSET -4
 844        jae nmi_stack_correct
 845        cmpl $ia32_sysenter_target,12(%esp)
 846        je nmi_debug_stack_check
 847nmi_stack_correct:
 848        /* We have a RING0_INT_FRAME here */
 849        pushl %eax
 850        CFI_ADJUST_CFA_OFFSET 4
 851        SAVE_ALL
 852        TRACE_IRQS_OFF
 853        xorl %edx,%edx          # zero error code
 854        movl %esp,%eax          # pt_regs pointer
 855        call do_nmi
 856        jmp restore_nocheck_notrace
 857        CFI_ENDPROC
 858
 859nmi_stack_fixup:
 860        RING0_INT_FRAME
 861        FIX_STACK(12,nmi_stack_correct, 1)
 862        jmp nmi_stack_correct
 863
 864nmi_debug_stack_check:
 865        /* We have a RING0_INT_FRAME here */
 866        cmpw $__KERNEL_CS,16(%esp)
 867        jne nmi_stack_correct
 868        cmpl $debug,(%esp)
 869        jb nmi_stack_correct
 870        cmpl $debug_esp_fix_insn,(%esp)
 871        ja nmi_stack_correct
 872        FIX_STACK(24,nmi_stack_correct, 1)
 873        jmp nmi_stack_correct
 874
 875nmi_espfix_stack:
 876        /* We have a RING0_INT_FRAME here.
 877         *
 878         * create the pointer to lss back
 879         */
 880        pushl %ss
 881        CFI_ADJUST_CFA_OFFSET 4
 882        pushl %esp
 883        CFI_ADJUST_CFA_OFFSET 4
 884        addw $4, (%esp)
 885        /* copy the iret frame of 12 bytes */
 886        .rept 3
 887        pushl 16(%esp)
 888        CFI_ADJUST_CFA_OFFSET 4
 889        .endr
 890        pushl %eax
 891        CFI_ADJUST_CFA_OFFSET 4
 892        SAVE_ALL
 893        TRACE_IRQS_OFF
 894        FIXUP_ESPFIX_STACK              # %eax == %esp
 895        xorl %edx,%edx                  # zero error code
 896        call do_nmi
```

```
 897         RESTORE_REGS
 898         lss 12+4(%esp), %esp            # back to espfix stack
 899         CFI_ADJUST_CFA_OFFSET -24
 900         jmp irq_return
 901         CFI_ENDPROC
 902KPROBE_END(nmi)
 903
 904#ifdef CONFIG_PARAVIRT
 905ENTRY(native_iret)
 906         iret
 907.section __ex_table,"a"
 908         .align 4
 909         .long native_iret, iret_exc
 910.previous
 911END(native_iret)
 912
 913ENTRY(native_irq_enable_sysexit)
 914         sti
 915         sysexit
 916END(native_irq_enable_sysexit)
 917#endif
 918
 919KPROBE_ENTRY(int3)
 920         RING0_INT_FRAME
 921         pushl $-1                       # mark this as an int
 922         CFI_ADJUST_CFA_OFFSET 4
 923         SAVE_ALL
 924         TRACE_IRQS_OFF
 925         xorl %edx,%edx          # zero error code
 926         movl %esp,%eax          # pt_regs pointer
 927         call do_int3
 928         jmp ret_from_exception
 929         CFI_ENDPROC
 930KPROBE_END(int3)
 931
 932ENTRY(overflow)
 933         RING0_INT_FRAME
 934         pushl $0
 935         CFI_ADJUST_CFA_OFFSET 4
 936         pushl $do_overflow
 937         CFI_ADJUST_CFA_OFFSET 4
 938         jmp error_code
 939         CFI_ENDPROC
 940END(overflow)
 941
 942ENTRY(bounds)
 943         RING0_INT_FRAME
 944         pushl $0
 945         CFI_ADJUST_CFA_OFFSET 4
 946         pushl $do_bounds
 947         CFI_ADJUST_CFA_OFFSET 4
 948         jmp error_code
 949         CFI_ENDPROC
 950END(bounds)
 951
 952ENTRY(invalid_op)
 953         RING0_INT_FRAME
 954         pushl $0
 955         CFI_ADJUST_CFA_OFFSET 4
 956         pushl $do_invalid_op
 957         CFI_ADJUST_CFA_OFFSET 4
```

```
 958         jmp error_code
 959         CFI_ENDPROC
 960END(invalid_op)
 961
 962ENTRY(coprocessor_segment_overrun)
 963         RING0_INT_FRAME
 964         pushl $0
 965         CFI_ADJUST_CFA_OFFSET 4
 966         pushl $do_coprocessor_segment_overrun
 967         CFI_ADJUST_CFA_OFFSET 4
 968         jmp error_code
 969         CFI_ENDPROC
 970END(coprocessor_segment_overrun)
 971
 972ENTRY(invalid_TSS)
 973         RING0_EC_FRAME
 974         pushl $do_invalid_TSS
 975         CFI_ADJUST_CFA_OFFSET 4
 976         jmp error_code
 977         CFI_ENDPROC
 978END(invalid_TSS)
 979
 980ENTRY(segment_not_present)
 981         RING0_EC_FRAME
 982         pushl $do_segment_not_present
 983         CFI_ADJUST_CFA_OFFSET 4
 984         jmp error_code
 985         CFI_ENDPROC
 986END(segment_not_present)
 987
 988ENTRY(stack_segment)
 989         RING0_EC_FRAME
 990         pushl $do_stack_segment
 991         CFI_ADJUST_CFA_OFFSET 4
 992         jmp error_code
 993         CFI_ENDPROC
 994END(stack_segment)
 995
 996KPROBE_ENTRY(general_protection)
 997         RING0_EC_FRAME
 998         pushl $do_general_protection
 999         CFI_ADJUST_CFA_OFFSET 4
1000         jmp error_code
1001         CFI_ENDPROC
1002KPROBE_END(general_protection)
1003
1004ENTRY(alignment_check)
1005         RING0_EC_FRAME
1006         pushl $do_alignment_check
1007         CFI_ADJUST_CFA_OFFSET 4
1008         jmp error_code
1009         CFI_ENDPROC
1010END(alignment_check)
1011
1012ENTRY(divide_error)
1013         RING0_INT_FRAME
1014         pushl $0                        # no error code
1015         CFI_ADJUST_CFA_OFFSET 4
1016         pushl $do_divide_error
1017         CFI_ADJUST_CFA_OFFSET 4
1018         jmp error_code
```

```
1019        CFI_ENDPROC
1020END(divide_error)
1021
1022#ifdef CONFIG_X86_MCE
1023ENTRY(machine_check)
1024        RING0_INT_FRAME
1025        pushl $0
1026        CFI_ADJUST_CFA_OFFSET 4
1027        pushl machine_check_vector
1028        CFI_ADJUST_CFA_OFFSET 4
1029        jmp error_code
1030        CFI_ENDPROC
1031END(machine_check)
1032#endif
1033
1034ENTRY(spurious_interrupt_bug)
1035        RING0_INT_FRAME
1036        pushl $0
1037        CFI_ADJUST_CFA_OFFSET 4
1038        pushl $do_spurious_interrupt_bug
1039        CFI_ADJUST_CFA_OFFSET 4
1040        jmp error_code
1041        CFI_ENDPROC
1042END(spurious_interrupt_bug)
1043
1044ENTRY(kernel_thread_helper)
1045        pushl $0                # fake return address for unwinder
1046        CFI_STARTPROC
1047        movl %edx,%eax
1048        push %edx
1049        CFI_ADJUST_CFA_OFFSET 4
1050        call *%ebx
1051        push %eax
1052        CFI_ADJUST_CFA_OFFSET 4
1053        call do_exit
1054        CFI_ENDPROC
1055ENDPROC(kernel_thread_helper)
1056
1057#ifdef CONFIG_XEN
1058/* Xen doesn't set %esp to be precisely what the normal sysenter
1059   entrypoint expects, so fix it up before using the normal path. */
1060ENTRY(xen_sysenter_target)
1061        RING0_INT_FRAME
1062        addl $5*4, %esp         /* remove xen-provided frame */
1063        CFI_ADJUST_CFA_OFFSET -5*4
1064        jmp sysenter_past_esp
1065        CFI_ENDPROC
1066
1067ENTRY(xen_hypervisor_callback)
1068        CFI_STARTPROC
1069        pushl $0
1070        CFI_ADJUST_CFA_OFFSET 4
1071        SAVE_ALL
1072        TRACE_IRQS_OFF
1073
1074        /* Check to see if we got the event in the critical
1075           region in xen_iret_direct, after we've reenabled
1076           events and checked for pending events.  This simulates
1077           iret instruction's behaviour where it delivers a
1078           pending interrupt when enabling interrupts. */
1079        movl PT_EIP(%esp),%eax
```

```
1080        cmpl $xen_iret_start_crit,%eax
1081        jb   1f
1082        cmpl $xen_iret_end_crit,%eax
1083        jae  1f
1084
1085        jmp  xen_iret_crit_fixup
1086
1087ENTRY(xen_do_upcall)
10881:      mov %esp, %eax
1089        call xen_evtchn_do_upcall
1090        jmp  ret_from_intr
1091        CFI_ENDPROC
1092ENDPROC(xen_hypervisor_callback)
1093
1094# Hypervisor uses this for application faults while it executes.
1095# We get here for two reasons:
1096#  1. Fault while reloading DS, ES, FS or GS
1097#  2. Fault while executing IRET
1098# Category 1 we fix up by reattempting the load, and zeroing the
segment
1099# register if the load fails.
1100# Category 2 we fix up by jumping to do_iret_error. We cannot use the
1101# normal Linux return path in this case because if we use the IRET
hypercall
1102# to pop the stack frame we end up in an infinite loop of failsafe
callbacks.
1103# We distinguish between categories by maintaining a status value in
EAX.
1104ENTRY(xen_failsafe_callback)
1105        CFI_STARTPROC
1106        pushl %eax
1107        CFI_ADJUST_CFA_OFFSET 4
1108        movl $1,%eax
11091:      mov 4(%esp),%ds
11102:      mov 8(%esp),%es
11113:      mov 12(%esp),%fs
11124:      mov 16(%esp),%gs
1113        testl %eax,%eax
1114        popl %eax
1115        CFI_ADJUST_CFA_OFFSET -4
1116        lea 16(%esp),%esp
1117        CFI_ADJUST_CFA_OFFSET -16
1118        jz 5f
1119        addl $16,%esp
1120        jmp iret_exc            # EAX != 0 => Category 2 (Bad IRET)
11215:      pushl $0                # EAX == 0 => Category 1 (Bad segment)
1122        CFI_ADJUST_CFA_OFFSET 4
1123        SAVE_ALL
1124        jmp ret_from_exception
1125        CFI_ENDPROC
1126
1127.section .fixup,"ax"
11286:      xorl %eax,%eax
1129        movl %eax,4(%esp)
1130        jmp 1b
11317:      xorl %eax,%eax
1132        movl %eax,8(%esp)
1133        jmp 2b
11348:      xorl %eax,%eax
1135        movl %eax,12(%esp)
1136        jmp 3b
```

```
11379:      xorl %eax,%eax
1138        movl %eax,16(%esp)
1139        jmp 4b
1140.previous
1141.section __ex_table,"a"
1142        .align 4
1143        .long 1b,6b
1144        .long 2b,7b
1145        .long 3b,8b
1146        .long 4b,9b
1147.previous
1148ENDPROC(xen_failsafe_callback)
1149
1150#endif  /* CONFIG_XEN */
1151
1152#ifdef CONFIG_FUNCTION_TRACER
1153#ifdef CONFIG_DYNAMIC_FTRACE
1154
1155ENTRY(mcount)
1156        ret
1157END(mcount)
1158
1159ENTRY(ftrace_caller)
1160        pushl %eax
1161        pushl %ecx
1162        pushl %edx
1163        movl 0xc(%esp), %eax
1164        movl 0x4(%ebp), %edx
1165        subl $MCOUNT_INSN_SIZE, %eax
1166
1167.globl ftrace_call
1168ftrace_call:
1169        call ftrace_stub
1170
1171        popl %edx
1172        popl %ecx
1173        popl %eax
1174
1175.globl ftrace_stub
1176ftrace_stub:
1177        ret
1178END(ftrace_caller)
1179
1180#else /* ! CONFIG_DYNAMIC_FTRACE */
1181
1182ENTRY(mcount)
1183        cmpl $ftrace_stub, ftrace_trace_function
1184        jnz trace
1185.globl ftrace_stub
1186ftrace_stub:
1187        ret
1188
1189        /* taken from glibc */
1190trace:
1191        pushl %eax
1192        pushl %ecx
1193        pushl %edx
1194        movl 0xc(%esp), %eax
1195        movl 0x4(%ebp), %edx
1196        subl $MCOUNT_INSN_SIZE, %eax
1197
```

```
1198        call *ftrace_trace_function
1199
1200        popl %edx
1201        popl %ecx
1202        popl %eax
1203
1204        jmp ftrace_stub
1205END(mcount)
1206#endif /* CONFIG_DYNAMIC_FTRACE */
1207#endif /* CONFIG_FUNCTION_TRACER */
1208
1209.section .rodata,"a"
1210#include "syscall_table_32.S"
1211
1212syscall_table_size=(.-sys_call_table)
1213
```