Chiamate di sistema

`int_128.s` 4547 `isr_syscall.c` 4549 `syscall.c` 4547 `syscall.h` 4548 `vsyscall.c` 4549

Nel sistema in corso di realizzazione sono previste le chiamate di sistema, anche se in pratica sono inutili, dal momento che non è possibile gestire processi elaborativi indipendenti. Queste chiamate si ottengono mettendo gli argomenti nella pila e utilizzando l'interruzione 128 (ovvero 80_{16}). Si osservi che questo meccanismo è diverso da quello usato dal kernel Linux, dove gli argomenti sono passati normalmente attraverso i registri del microprocessore.

Il punto di inizio per una chiamata di sistema è la funzione ***syscall()***, con la quale va indicato il numero della chiamata, seguito dagli argomenti necessari, in base al contesto.

Listato u172.1. '`./05/lib/sys/syscall.c`'

```
#include <sys/syscall.h>
#include <kernel/int.h>
uint32_t
syscall (int n, ...)
{
    return int_128 ();
}
```

Come si vede, ci si limita a utilizzare la funzione ***int_128()***, scritta però in linguaggio assemblatore, come si vede nel listato successivo.

Listato u172.2. ‘./05/lib/int/int_128.s’

```
.globl int_128
#
int_128:
    int $128
    ret
```

Questa doppia mediazione ha delle conseguenze nella composizione della pila dei dati, al momento dell’avvio della funzione che deve trattare l’interruzione.

File di intestazione «syscall.h»

«

Il file di intestazione ‘syscall.h’ dichiara le funzioni usate per generare una chiamata di sistema e poi per eseguirla; inoltre, si definiscono delle macro-variabili per dare un nome alle chiamate che in realtà sono indicate solo per numero.

Listato u172.3. ‘./05/include/sys/syscall.h’

```
#ifndef _SYSCALL_H
#define _SYSCALL_H      1

#include <inttypes.h>
#include <stdarg.h>

#define SYSCALL_malloc       1
#define SYSCALL_realloc      2
#define SYSCALL_free         3
#define SYSCALL_console_putc 4

uint32_t syscall  (int n, ...);
uint32_t vsyscall (int n, va_list ap);

#endif
```

Fasi successive all'interruzione

Una volta provocata l'interruzione 128, si ottiene l'attivazione della funzione ***isr_128()***, la quale avvia a sua volta la funzione ***isr_syscall()*** che deve provvedere a ripescare gli argomenti della chiamata originale, quindi avvia la funzione che può elaborarli: ***vsyscall()***.

Listato u172.4. '`./05/lib/int/isr_syscall.c`'

```
#include <kernel/int.h>
#include <sys/syscall.h>
uint32_t
isr_syscall (uint32_t start, ...)
{
    va_list ap;
    uint32_t value;
    //
    // Colloca il puntatore all'inizio.
    //
    va_start (ap, start);
    //
    // Salta i dati che non servono.
    //
    value = va_arg (ap, uint32_t); // CS
    value = va_arg (ap, uint32_t); // EFLAGS
    value = va_arg (ap, uint32_t); // ???
    value = va_arg (ap, uint32_t); // ESP
    value = va_arg (ap, uint32_t); // SS
    value = va_arg (ap, uint32_t); // EIP
    value = va_arg (ap, uint32_t); // EIP
    value = va_arg (ap, uint32_t); // n. chiamata
    //
    // Attualmente «ap» punta all'argomento successivo
    // al numero di chiamata.
```

```
    //
    return vsyscall (value, ap);
}
```

Listato u172.5. ‘./05/lib/sys/vsyscall.c’

```
#include <sys/syscall.h>
#include <stdint.h>
#include <inttypes.h>
#include <stdlib.h>
#include <stdarg.h>
#include <kernel/vga.h>

uint32_t
vsyscall (int n, va_list ap)
{
    if (n == SYSCALL_malloc)
      {
        size_t  size = va_arg (ap, size_t);
        return (uint32_t) malloc (size);
      }
    else if (n == SYSCALL_realloc)
      {
        void   *ptr  = va_arg (ap, void*);   // Qui, «void*» va scritto
        size_t  size = va_arg (ap, size_t);  // attaccato e senza parentesi.
        return (uint32_t) realloc (ptr, size);
      }
    else if (n == SYSCALL_free)
      {
        void   *ptr  = va_arg (ap, void*);
        free (ptr);
        return 0;
      }
    else if (n == SYSCALL_console_putc)
      {
        int     c   = va_arg (ap, int);
        vga_putc (c);
        return (uint32_t) c;
      }
    else
      {
        printf ("[%s] ERROR: unknown syscall: %i!\n", __func__, n);
```

```
        return -1;
      }
}
```

Verifica del funzionamento

Per verificare il funzionamento delle chiamate di sistema, si può modificare il file ‘`kernel_main.c`’ nel modo seguente, allo scopo di visualizzare sullo schermo la parola «Ciao».

Figura u172.6. Modifiche da apportare al file ‘`./05/kernel/kernel_main.c`’

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>
#include <sys/syscall.h>
…
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        //
        // Omissis.
        //
        //
        // Predispone la tabella IDT.
        //
        idt();
        //
        // Prova le chiamate di sistema.
        //
        syscall (SYSCALL_console_putc, 'C');
        syscall (SYSCALL_console_putc, 'i');
        syscall (SYSCALL_console_putc, 'a');
        syscall (SYSCALL_console_putc, 'o');
```

```
        syscall (SYSCALL_console_putc, '\n');
…
```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove prima della conclusione si vede l'emissione della parola «Ciao»:

```
05 20070821155848
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010DADC  avail. 0010DADC..01EF0000
[kernel_memory_show] text   00100000..00104BA8  rodata 00104BC0..00105144
[kernel_memory_show] data   00105144..00105144  bss    00105160..0010DADC
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010D1A8 limit: 0x0017
[gdt_print] 0 0000000000000000000000000000000 0000000010000000001000000000000
[gdt_print] 1 0000000000000000001111011110000 0000000011000000100110100000000
[gdt_print] 2 0000000000000000001111011110000 0000000011000000100100100000000
[mm_init] available memory: 31335712 byte
[irq_remap] PIC (programmable interrupt controller) remap: ICW1, ICW2, ICW3,
ICW4, OCW1.
Ciao
[kernel_main] system halted
```