

# Lezione E10

1/12/2022  
(ero assente)

## Scheduler per job interrompibili

Sistemi embedded e real-time

1 dicembre 2022

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

Marco Cesati

Dipartimento di Ingegneria Civile e Ingegneria Informatica  
Università degli Studi di Roma Tor Vergata

SERT22

E10.1

## Di cosa parliamo in questa lezione?

In questa lezione scriviamo il codice dello scheduler per job interrompibili

- ➊ Il cambio di contesto
- ➋ Creazione di un nuovo task
- ➌ Scheduler (prima versione)
- ➍ Modifiche alla funzione `_irq_handler()`
- ➎ La funzione `_irq_schedule()`
- ➏ La funzione `_sys_schedule()`
- ➐ Scheduler (seconda versione)

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.2

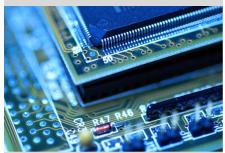
## Contesto prima dell'esecuzione di `_bsp_irq()`

- La CPU è in modalità SYSTEM
- I registri AACS-clobbered del flusso di esecuzione interrotto salvati sullo stack sono `r0-r3, r12, r14 (=lr)`
- Nel registro `r12` è caricato l'indirizzo  $\Omega$  del gestore di medio livello dell'interruzione `_bsp_irq()`

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	r13	sp	lr	pc	cpsr	spsr
SYS IRQ	N-32	Q	C	D	E	F	G	H	I	J	K	L	$\Omega$	N-32	#	#	*	///	
STK	(A)	B	C	D	M	O	P+4	Q	X	Y	Z		A	B			Q		

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.3

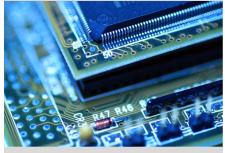
## Contesto dopo la terminazione di `_bsp_irq()`

- La CPU è in modalità SYSTEM
- In cima allo stack vi sono i valori di alcuni registri del flusso di esecuzione da recuperare:
  - Registri AACS-clobbered `r0-r3, r12, r14 (=lr)`
  - Registro `r15 (=pc)`, ossia l'indirizzo  $P + 4$  dell'istruzione successiva del flusso di esecuzione da recuperare
  - Registro `cpsr` (valore `Q`)

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	r13	sp	lr	pc	cpsr	spsr
SYS IRQ	?	?	?	?	E	F	G	H	I	J	K	L	?	N-32	?	#	*	///	
STK	(A)	B	C	D	M	O	P+4	Q	X	Y	Z		?	?			?		

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.4

## Cambio di contesto

Scheduler per job interrompibili

Marco Cesati



- Ogni job ha un proprio stack
- I valori dei registri r0–r3, r12, r14, r15 e cpsr sono salvati sul proprio stack
- È necessario salvare altrove i valori dei registri r4–r11 e r13 (=sp)
  - Salvataggio nel descrittore del task!
- Effettuare il cambio di contesto significa modificare sp in modo che faccia riferimento allo stack di un task differente
- Il cambio di contesto viene effettuato subito dopo la terminazione di `_bsp_irq()`
  - Le istruzioni macchina seguenti in `_irq_handler()` recuperano il contesto di un flusso di esecuzione (job) diverso da quello interrotto

Il cambio di contesto avviene solo se si sta per tornare all'esecuzione di un job (attenzione alle interruzioni annidate)

SERT22

E10.5

## Stack specifico per ogni task

Scheduler per job interrompibili

Marco Cesati



- Gli stack sono allocati nella sezione `.stack` in un unico vettore
- Ogni stack ha lunghezza predefinita
- Ogni stack è allineato a pagine di memoria
- Gli stack crescono per indirizzi decrescenti

```
#define STACK_SIZE 4096
char stacks [MAX_NUM_TASKS*STACK_SIZE]
__attribute__((aligned (STACK_SIZE),
                section (.stack)));
const char *stack0_top =
    stacks + MAX_NUM_TASKS*STACK_SIZE;
```

Successivamente inizializziamo il registro sp alla cima dello stack del task 0.

In tal modo, quando compiliamo, ci vuole molto più tempo: per la sezione "stack", si ha un'inizializzazione esplicita a 0 da parte del compilatore. Evitabile facendo in modo che gli stack non finiscano in sert.bin

SERT22

E10.6

## Il descrittore del task

Scheduler per job interrompibili

Marco Cesati

```
struct task {
    int valid;
    job_t job;
    void *arg;
    unsigned long releasetime;
    int released;
    int period;
    int priority;
    const char *name;
    u32 sp;
    u32 regs[8];
};
```



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

- La variabile `current` contiene l'indirizzo del task in esecuzione
- Se nessun job è eseguibile, `current` punta al descrittore del task con TID 0 (*idle task*)

SERT22

E10.7

## La funzione \_switch\_to()

Scheduler per job interrompibili

Marco Cesati

```
void _switch_to(struct task *)
__attribute__((naked));
```

```
void _switch_to(struct task *to)
{
    save_regs(current->regs);
    load_regs(to->regs);
    switch_tasks(current, to);
    current = to;
    naked_return();
}
```



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.8

## Funzioni “naked”

Scheduler per job interrompibili

Marco Cesati



- L'attributo `naked` è una estensione del compilatore `gcc` per l'architettura ARM
- Forza il compilatore a *non* emettere codice per il prologo e l'epilogo della funzione
- Nessuna istruzione generata automaticamente per salvare il contenuto di registri sullo stack
- Tuttavia non viene nemmeno emessa l'istruzione di “return” per terminare la funzione
  - È necessario generare esplicitamente l'istruzione di “return”

```
#define naked_return() \
    __asm__ __volatile__ ("bx lr")
```

SERT22

E10.9

## Salvataggio e ripristino dei registri r4–r11

Scheduler per job interrompibili

Marco Cesati



Per salvare il contenuto dei registri r4–r11:

```
#define save_regs(regs) \
    __asm__ __volatile__ ("stmia %0,{r4-r11}" \
                         : : "r" (regs) : "memory")
```

SERT22

E10.10

Per ripristinare il contenuto degli stessi registri:

```
#define load_regs(regs) \
    __asm__ __volatile__ ("ldmia %0,{r4-r11}" \
                         : : "r" (regs) : "r4", "r5", "r6", \
                           "r7", "r8", "r9", "r10", "r11", "memory")
```

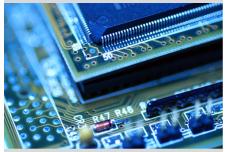
SERT22

E10.10

## La macro switch\_task

Scheduler per job interrompibili

Marco Cesati



La macro `switch_task` salva e ripristina il contenuto del registro r13 (=sp)

```
#define switch_task(from, to) \
    __asm__ __volatile__ ("str sp,%0\n\t" \
                          "ldr sp,%1\n\t" \
                           : = "m" ((from)->sp), "m" ((to)->sp) \
                           : "sp", "memory")
```

Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.11

## Inizializzazione del contesto di un nuovo task

Scheduler per job interrompibili

Marco Cesati



- Determinare la posizione dello stack del task
- Definire l'entry point del job
  - Non coincide con la funzione da eseguire nel job
- Inizializzare il contesto salvato nel descrittore del task
- Inizializzare lo stack del task
  - Deve avere la stessa struttura dello stack di un job interrotto da una interruzione

Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

STK	r0	r1	r2	r3	r12	lr	pc	spsr
-----	----	----	----	----	-----	----	----	------

N +4 +8 +12 +16 +20 +24 +28

SERT22

E10.12

## La funzione init\_task\_context()

```
void init_task_context(struct task *t,
                      int ntask) {
    int i;
    unsigned long *sp = (unsigned long *)
        (stack0_top - ntask * STACK_SIZE);
    /* spsr */
    *(--sp) = SYS_MODE;
    /* pc */
    *(--sp) = (unsigned long)
        task_entry_point;
    /* lr */
    *(--sp) = 0UL;
    /* r12 */
    *(--sp) = 0UL;
    /* r3 */
    *(--sp) = 0UL;
    /* r2 */
    *(--sp) = 0UL;
    /* r1 */
    *(--sp) = 0UL;
    /* r0 */
    *(--sp) = (unsigned long) t;
    t->sp = (unsigned long) sp;
    for (i = 0; i < 8; ++i)
        t->regs[i] = 0UL; /*r4-r11*/
}
```

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

## La funzione task\_entry\_point()

task\_entry\_point() è il punto di ingresso di ogni task

L'argomento (in r0) è l'indirizzo del descrittore di task

```
void task_entry_point(struct task *)
    __attribute__ ((naked));
void task_entry_point(struct task *t)
{
    for (;;) {
        if (!t->valid || !t->released)
            panic0();
        irq_enable();
        t->job(t->arg);
        irq_disable();
        --t->released;
        _sys_schedule();
    }
}
```

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

## La funzione task\_entry\_point() (2)

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

- Esegue un ciclo senza fine
  - Una iterazione per ciascun rilascio di un job del task
- La funzione del job è eseguita con interruzioni abilitate
- Per il resto il ciclo è eseguito con interruzioni disabilitate
- Quando un job termina si decrementa  $t \rightarrow released$ 
  - È il numero di job del task rilasciati e non completati
- La funzione `_sys_schedule()` seleziona un nuovo task da eseguire quando un job del task termina
  - È eseguita in modalità **SYSTEM**
  - La funzione “ritornerà” solo quando questo task verrà nuovamente selezionato dallo scheduler per l'esecuzione

SERT22

E10.15

## La funzione create\_task()

`create_task()` deve “saltare” il **task 0 (idle task)**

```
int create_task(job_t job, void *arg,
                int period, int delay,
                int priority, const char *name)
{
    int i;
    struct task *t;
    for (i=1 ; i<MAX_NUM_TASKS; ++i)
        if (!taskset[i].valid)
            break;
[...]
    irq_disable();
    ++num_tasks;
    init_task_context(t, i);
    t->valid = 1;
    irq_enable();
[...]
}
```

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.16

## Lo scheduler dei job

Costituito da diverse procedure:

- La funzione `check_periodic_tasks()`
  - Eseguita ad ogni tick in modalità **IRQ**
  - Controlla i rilasci dei job dei task periodici
- La funzione `schedule()`:
  - Eseguita in modalità **SYSTEM**
  - Scansiona il vettore di task per selezionare il job rilasciato di priorità massima

Chi invoca lo scheduler:

- La funzione Assembly `_irq_handler()`:
  - Alla conclusione della gestione dell'interruzione, esegue se necessario `schedule()`
- La funzione Assembly `_sys_schedule()`:
  - Invoca `schedule()` dalla modalità **SYSTEM**
  - Necessaria per gestire la terminazione di un job

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.17

## La funzione `check_periodic_tasks()`

```
void check_periodic_tasks(void) {
    unsigned long now = ticks;
    struct task *f;
    int i;
    for (i=0, f=taskset+1; i<num_tasks; ++f) {
        if (f - taskset > MAX_NUM_TASKS)
            panic0();
        if (!f->valid)
            continue;
        if (time_after_eq(now, f->releasetime)) {
            ++f->released;
            f->releasetime += f->period;
            trigger_schedule = 1;
            ++globalreleases;
        }
        ++i;
    }
}
```

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.18

## La funzione schedule() – 1<sup>a</sup> versione

schedule() restituisce l'indirizzo del descrittore del task da eseguire, oppure NULL se il task corrente è già il migliore

Scheduler per job interrompibili

Marco Cesati



```
struct task *schedule(void)
{
    struct task *best;
    unsigned long oldreleases;
    do {
        oldreleases = globalreleases;
        irq_enable();
        best = select_best_task();
        irq_disable();
    } while (oldreleases!=globalreleases);
    trigger_schedule = 0;
    best = (best!=current ? best : NULL);
    return best;
}
```

Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.19

## La funzione select\_best\_task()

Scheduler per job interrompibili

Marco Cesati



```
static inline
struct task *select_best_task(void) {
    int maxprio, i;
    struct task *best, *f;
    maxprio = MAXUINT;
    best = &taskset[0];
    for (i=0, f=taskset+1; i<num_tasks; ++f) {
        if (f - taskset >= MAX_NUM_TASKS)
            panic0();
        if (!f->valid)
            continue;
        ++i;
        if (!f->released)
            continue;
        if (f->priority < maxprio)
            maxprio = f->priority, best = f;
    }
    return best;
}
```

Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.19

## Modifica alla funzione \_irq\_handler() (1)

\_irq\_handler() deve consentire di invocare schedule() se necessario

```
[...]
bx      r12 /* call to _bsp_irq() */
msr    cpsr_c, #(SYS_MODE | NO_INT)
mov    r0, sp
add    sp, sp, #(8*4)
msr    cpsr_c, #(IRQ_MODE | NO_INT)
mov    sp, r0
ldr    r0, [sp, #(7*4)]
msr    spsr_cxsf, r0

: Nuove istruzioni :
ldmfd  sp, {r0-r3, r12, lr}^
nop
ldr    lr, [sp, #(6*4)]
movs   pc, lr
```

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

## Modifica alla funzione \_irq\_handler() (2)

Non si deve invocare schedule() se

- il flusso interrotto non è in modalità SYSTEM
- trigger\_schedule è uguale a zero

```
msr    spsr_cxsf, r0
ldr    r0,=irq_level
ldr    r0, [r0]
tst    r0, r0
bne   .Lnosched
ldr    r0,=trigger_schedule
ldr    r0, [r0]
tst    r0, r0
beq   .Lnosched
[...]
.Lnosched:
ldmfd  sp, {r0-r3, r12, lr}^
```

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

## Modifica alla funzione \_irq\_handler() (3)

Per invocare `schedule()` in modalità **SYSTEM** si termina l'esecuzione della interruzione forzando l'indirizzo di una procedura alternativa nel link register `lr`



Scheduler per job interrompibili

Marco Cesati

Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

```
msr      spsr_cxsfs, r0
ldr      r0, =irq_level
ldr      r0, [r0]
tst      r0, r0
bne      .Lnosched
ldr      r0, =trigger_schedule
ldr      r0, [r0]
tst      r0, r0
beq      .Lnosched
msr      spsr_c, #(SYS_MODE | NO_IRQ)
ldr      lr, =_irq_schedule
movs    pc, lr
.Lnosched:
ldmfd   sp, {r0-r3, r12, lr} ^
```

SERT22 E10.23

## Contesto prima della esecuzione di \_irq\_schedule()



Scheduler per job interrompibili

Marco Cesati

Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp	lr	pc	cpsr	spsr
SYS	?	?	?	?	E	F	G	H	I	J	K	L	?	N	?	#	*	///
IRQ	A	B	C	D	M	O	P+4	Q	X	Y	Z		?	?			?	
STK																		

SERT22 E10.24

## La funzione \_irq\_schedule() (1)

- Il valore restituito da `schedule()` è nel registro `r0`
- Se `r0` è nullo non occorre effettuare un cambio di contesto

Scheduler per job interrompibili

Marco Cesati



`_irq_schedule:`

```
sub    sp, sp, #32
ldr    r12, =schedule
mov    lr, pc
bx    r12
tst    r0, r0
beq    .Lnoswitch
ldr    r12, =_switch_to
mov    lr, pc
bx    r12
.Lnoswitch:
[ ... ]
```

Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

Al termine di `_switch_to` lo stack è stato cambiato: `sp` punta ora alla cima dello stack del task che deve essere eseguito

SERT22

E10.25

## La funzione \_irq\_schedule() (2)

- Legge dallo stack il valore corrispondente al registro `spsr`
- Lo scrive direttamente entro `cpsr`
  - Le istruzioni successive non modificano `cpsr`
- Ripristina i valori dei registri `r0–r3`, `r12` e `lr`
  - Contemporaneamente aggiunge 32 al valore di `sp`

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

```
[ ... ]
.Lnoswitch:

```

```
ldr    r0, [sp, #(7*4)]
msr    cpsr_csf, r0
ldmfd sp!, {r0-r3, r12, lr}
```

```
[ ... ]
```

SERT22

E10.26

## La funzione `_irq_schedule()` (3)

La situazione del contesto dopo `ldmfd`:

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	sp	lr	pc		
SYS	A	B	C	D	E	F	G	H	I	J	K	L	M	N-8	O	#	Q	///
IRQ	P+4	Q	X	Y	Z									?	?		?	

È necessario:

- Saltare all'indirizzo `P+4` memorizzato in cima allo stack
- Bilanciare lo stack aggiungendo 8 al registro `sp`
- Evitare la modifica di altri registri oltre `sp` e `pc`

```
[ . . . ]
ldmfd    sp!, {r0-r3,r12,lr}
ldr      pc, [sp], #(2*4)
```

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.27

13/12/2022

## La funzione `_sys_schedule()` (1)

- Invocata in modalità SYSTEM
  - Lo stack non contiene i valori salvati da `_irq_handler()`
- È necessario salvare i registri AACSB-clobbered sullo stack
  - ...ma non è possibile utilizzare i registri della modalità IRQ

```
_sys_schedule:
str      lr, [sp, #- (4*2)]!
mrs      lr, cpsr
str      lr, [sp, #4]
ldr      lr, [sp]
[ . . . ]
```

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

`_irq_handler()`

`_irq_schedule()`

`_sys_schedule()`

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.28

- Il valore di ritorno in `lr` viene salvato sullo stack
  - Punta all'interno del ciclo in `task_entry_point()`
  - `sp` viene aggiornato in modo da lasciare una posizione libera sotto la cima
- `cpsr` viene salvato sullo stack nella posizione libera
- `lr` viene recuperato dallo stack (la cima non è modificata)

## La funzione \_sys\_schedule() (2)

- Si salvano sullo stack i registri r0–r3, r12 e lr
- Dopo il salvataggio la struttura dei valori sullo stack è esattamente identica a quella che si ha in seguito ad una interruzione
- La procedura termina eseguendo un salto entro la funzione \_irq\_schedule()

```
[ . . . ]
    ldr      lr, [sp]
    stmfd   sp!, {r0-r3, r12, lr}    aggiorno sp e mi
    b       .Lnosub32      riprendo i registri
                                salto al pezzo di codice che evoca "schedule()"
```

```
_irq_schedule:
    sub    sp, sp, #32
.Lnosub32:
    ldr    r12, =schedule
[ . . . ]
```

Con questo codice ci siamo "re-innestati" nella porzione di codice che chiama "schedule()" avendo tutti i registri salvati correttamente.

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

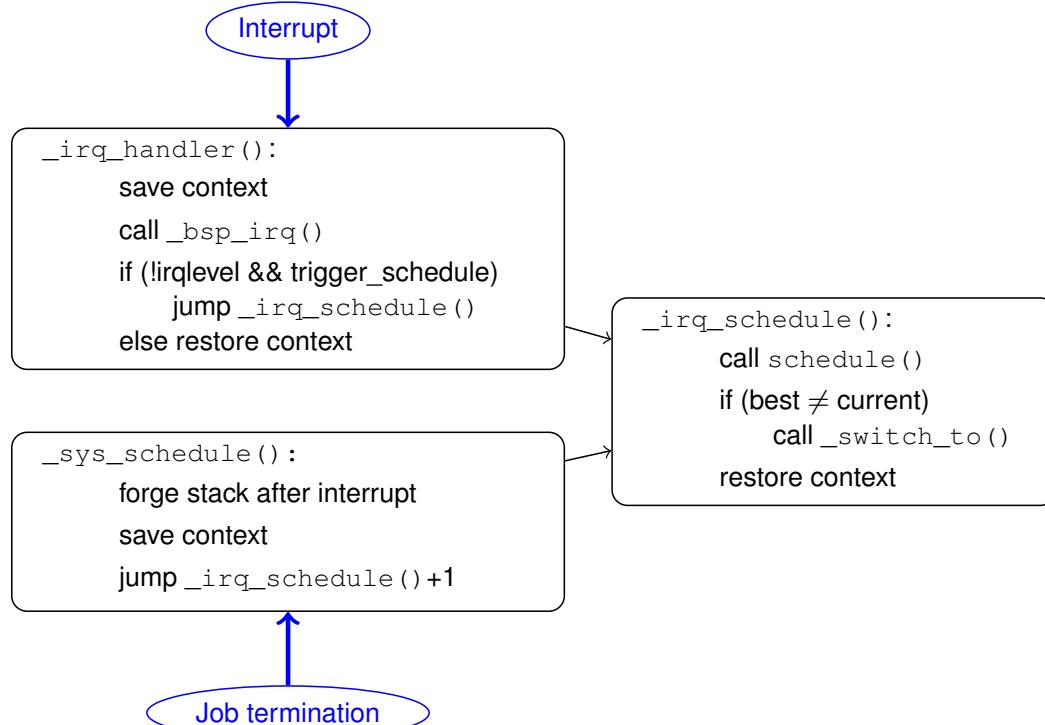
\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.29

## Schema riassuntivo



Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

SERT22

E10.30

## Race condition in schedule()

CONTESTO:

schedule() invoca "select\_best\_task()" con le interruzioni ABILITATE.  
Si tratta di una funzione piuttosto lenta, per cui non voglio eseguirla con gli interrupt disabilitati. Questo vuol dire che un interrupt può sopravvenire mentre seleziono il best task, e quindi questo porta a invocare nuovamente "schedule()", innestata rispetto allo schedule originale. "E' una funzione rientrante".

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

- Problema: la funzione `schedule()` è **rientrante**
  - Ad esempio: quando un job termina viene invocata
  - Durante la sua esecuzione avviene una interruzione di tick
- Disabilitare le interruzioni durante la scansione del vettore di task porterebbe a tempi di blocco eccessivi
- Utilizziamo come meccanismo di sincronizzazione una variabile "sentinella" `do_not_enter`

Essere una funzione rientrante, di per sè, non è un problema.

Il problema nasce se questi toccano VARIABILI GLOBALI, portando così ad una potenziale inconsistenza delle variabili stesse.

Questo ci porta ad una alla seguente modifica: impedire le invocazioni di `schedule()` innestati.

SERT22

E10.31

Questa qui presentata è l'implementazione finale.

PASSIAMO ALLA SCALETTA E11 per vedere più nel dettaglio.

## La funzione `schedule()` – versione finale

Scheduler per job interrompibili

Marco Cesati



Schema della lezione

Cambio di contesto

Creazione di un task

Scheduler (1<sup>a</sup> vers.)

\_irq\_handler()

\_irq\_schedule()

\_sys\_schedule()

Scheduler (2<sup>a</sup> vers.)

```
struct task *schedule(void) {
    static int do_not_enter = 0;
    struct task *best;
    unsigned long oldreleases;
    if (do_not_enter != 0)
        return NULL;
    do_not_enter = 1;
    do {
        ...
    } while (oldreleases != globalreleases);
    trigger_schedule = 0;
    do_not_enter = 0;
    best = (best != current ? best : NULL);
    return best;
}
```

SERT22

E10.32