SCALETTA LEZIONE SERT 1.12.2022 (E10)

1 Implementazione di stack separati per ogni task

1.1 Aggiungere una sezione ".stack" in sert.lds dopo la sezione ".bss"

```
[...]
   _bss_end = .;
} > ram
.stack : {
   . = ALIGN(4096);
   *(.stack)
} > ram
```

di tutto il processo

1.2 Modificare tasks.c per allocare gli stack nella sezione .stack

```
#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;
```

1.3 Modificare startup.S per caricare lo stack del task 0

```
[...]
ldr     sp,=stack0_top
ldr     sp,[sp]
b       _init
```

adesso è per tutti i task

1.4 Provare a caricare l'immagine: e' diventata molto piu' grande!

1.5 Modificare Makefile per evitare che la porzione degli stack finisca nell'immagine sert.bin (inutilmente)

non è una parte di codice, in quanto usato a runtime.

```
[...]
%.bin: %.elf
    $(OBJCOPY) -S -R .stack -O binary $< $@
```

1.6 Definire la variabile current che punta al descrittore di task in esecuzione:

```
struct task *current;           (in sched.c)
extern struct task *current     (in comm.h)
current=&taskset[0];            (in tasks.c, init_taskset())
```

1.7 Il task 0 non e' un vero task periodico: e' l'"idle" task che lo scheduler seleziona quando non esistono altri job eseguibili nel sistema

1.7.1 Si puo' considerare il task 0 come in esecuzione durante l'inizializzazione

2 Implementazione del cambio di contesto

2.1 Definizione del cambio di contesto:

2.1.1 Registri AAPCS-clobbered r0-r3,r12,r14/lr: salvati dal gestore di interruzione sullo stack attivo al momento della interruzione

2.1.2 Registri non AAPCS-clobbered r4-r11, r13/sp: da salvare nel descrittore di task

2.1.3 Tecnicamente il cambio di contesto si considera effettuato quando si modifica il valore di r13/sp per indirizzare lo stack del task che deve (ri)cominciare ad essere eseguito

2.2 Aggiungere al descrittore di task in comm.h i campi per il salvataggio del contesto d'esecuzione

```
struct task {
[...]
    unsigned long sp;
    unsigned long regs[8];

};
```

2.3 Scrivere la funzione "naked" _switch_to() in sched.c:

```
[...]

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

2.3.1 L'attributo "naked" (specifico per l'architettura ARM) forza il compilatore a non generare alcun codice per l'invocazione e terminazione della funzione

2.3.2 Scrivere la macro save_regs:

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

2.3.3 Scrivere la macro load_regs:

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

2.3.4 Scrivere la macro switch_task:

```
#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")
```

2.3.5 Scrivere la macro naked_return():

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

2.4 Controllare il codice prodotto per _switch_to tramite il

disassemblatore del cross-compiler (arm*objdump)

3 Inizializzazione del contesto d'esecuzione di un nuovo task
  3.1 Consiste nel:
    3.1.1 determinare lo stack per il task
    3.1.2 inizializzare lo stack per il task
    3.1.3 definire il punto d'inizio del programma del task
    3.1.4 inizializzare il contesto salvato nel descrittore
  3.2 Lo stack viene inizializzato in modo da essere analogo allo stack di un task che e' stato interrotto da una interruzione:

```
    +---+---+---+---+---+---+---+----+
STK |r0 |r1 |r2 |r3 |r12|r14|RET|spsr|
    +---+---+---+---+---+---+---+----+
     N   +4  +8  +12 +16 +20 +24 +28
```

  3.4 Scrivere la funzione init_task_context() in tasks.c

```
void init_task_context(struct task *t, int ntask)
{
   unsigned long *sp;
   int i;
   sp = (unsigned long *)(stack0_top - ntask * STACK_SIZE);
   *(--sp) = 0x1ful; /* SYS_MODE */  /* spsr */
   *(--sp) = (unsigned long)task_entry_point;   /* ret addr */
   *(--sp) = 0UL;     /* r14/lr */
   *(--sp) = 0UL;     /* r12 */
   *(--sp) = 0UL;     /* r3 */
   *(--sp) = 0UL;     /* r2 */
   *(--sp) = 0UL;     /* r1 */
   *(--sp) = (unsigned long)t;    /* r0 */
   t->sp = (unsigned long)sp;
   for (i = 0; i < 8; ++i)
      t->regs[i] = 0UL; /* r4-r11 */
}
```

"-" perchè sta risalendo

me lo creo come visto nello stack sopra.

  3.5 I valori associati all'indirizzo di ritorno (ret) ed al registro r0 corrispondono all'esecuzione della funzione task_entry_point() con argomento l'indirizzo del descrittore del task stesso
    3.5.1 I valori associati ai registri r1-r3 ed r12 sono posti a 0.
    3.5.2 Il valore associato al link register (lr) e' azzerato poiche' task_entry_point e' senza ritorno
    3.5.3 Il valore associato al registro SPSR dev'essere impostato in maniera tale che modalita' di esecuzione, flags e maschera degli interrupt del processore al momento del salto a task_entry_point siano corretti
      3.5.3.1 Il valore 0x1F corrisponde alla modalita' SYSTEM, con le interruzioni non mascherate

4 Definizione del punto d'ingresso del codice di un task
  4.1 Scrivere in tasks.c la funzione "naked" task_entry_point()

```
void task_entry_point(struct task *) __attribute__ ((naked));
void task_entry_point(struct task *t)
{
   for (;;) {
      if (t->valid == 0 || t->released == 0)
```

```
        panic0();
      irq_enable();
      t->job(t->arg);
      irq_disable();
      --t->released;
      _sys_schedule();
    }
}
```

4.2 La funzione esegue un ciclo senza fine: vi sara' una iterazione del ciclo per ciascun rilascio di un job del task

4.3 Il job viene eseguito con le interruzioni abilitate, ma nel resto della funzione le interruzioni sono disabilitate

4.4 Quando un job termina il campo t->released viene decrementato: questo campo conserva il medesimo significato che aveva con lo scheduler per job non interrompibili: indica il numero di job rilasciati e non ancora completati del task

4.5 La funzione _sys_schedule(), invocata in SYSTEM mode, permette di selezionare un nuovo task da eseguire quando un job del task termina

5 Modificare la funzione create_task()

5.1 Cominciare il ciclo per la ricerca di un descrittore libero saltando la posizione 0 (riservata all' "idle" task)

```
for (i=1; i<MAX_NUM_TASKS; ++i)
```

prima lavoravamo con tick!

5.2 Aggiungere prima di irq_disable() l'inizializzazione del contesto del task

```
init_task_context(t, i);
```

conseguenza di avere l'interrompibilità

6 Modificare la funzione check_periodic_tasks() invocata ad ogni tick:

```
volatile unsigned long trigger_schedule = 0;
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)
            _panic(); /* Should never happen */
        if (!f->valid)
            continue;
        if (time_after_eq(now, f->releasetime)) {
            f->releasetime += f->period;
            ++f->released;
            trigger_schedule = 1;                /* <<< */
            ++globalreleases;
        }
        ++i;
    }
}
```

6.1 Saltare il task 0: l'idle task non e' periodico
6.2 Aggiungere un flag global "trigger_schedule", necessario per forzare l'invocazione in modalita' SYSTEM della funzione dello scheduler

7 Modificare la funzione select_best_task() che seleziona il task periodico con job rilasciati di priorita' massima, ovvero il task 0 (idle task) se nessun task periodico e' eseguibile

```
static inline struct task *select_best_task(void)
{
    unsigned long maxprio;
    int 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)
            _panic();
        if (!f->valid)
            continue;
        ++i;
        if (f->released == 0)
            continue;
        if (f->priority < maxprio) {
            maxprio = f->priority;
            best = f;
        }
    }
    return best;
}
```

7.1 Questa funzione puo' essere eseguita con interruzioni abilitate, percio' puo' esistere una "race condition" con check_periodic_tasks() sui campi f->relesed dei task: del problema si occupa la funzione invocante select_best_task
7.2 Notare che la ricerca del task periodico inizia dal task con TID=1, mentre il task 0 e' selezionato solo se nessun altro task e' eseguibile

8 Sostituire la funzione run_periodic_tasks con la funzione schedule() per valutare se e' necessario selezionare un nuovo task da porre in esecuzione

```
struct task *schedule(void)
{
    struct task *best;
    unsigned long oldreleases;
    do {
        oldreleases = globalreleases;
        best = select_best_task();
    } while (oldreleases != globalreleases);
    trigger_schedule = 0;
```

select_best_task la invoco quando noto la presenza di un nuovo task, ovvero la condizione del while.

trigger_schedule è a 0 se ho fatto la selezione dei task, è a 1 alla fine di esecuzione di un task.

```
    return (best != current ? best : NULL);
}
```

8.1 La funzione usa select_best_task() per selezionare il vincitore
8.2 La funzione restituisce l'indirizzo del descrittore del vincitore, a meno che il vincitore non sia gia' in esecuzione, nel qual caso la funzione restituisce NULL
8.3 La funzione e' invocata in modalita' SYSTEM
8.4 Il meccanismo per invocare schedule() in modo asincrono consiste nell'impostare il valore 1 in trigger_schedule (funzione check_periodic_tasks())
8.5 Sostituire la definizione da run_periodic_tasks() a schedule() in comm.h

9 Modificare il gestore di interruzioni a basso livello _irq_handler() per invocare, se necessario, le funzioni schedule() e _switch_to()
9.1 Inseriamo la modifica nel percorso di "uscita" dall'interruzione, esattamente dopo essere tornati da _bsp_irq(): medio livello

```
ldr r12, = _bsp_irq
mov lr, pc
bx r12
msr cpsr_c, #(SYS_MODE|NO_INT)
```

9.2 Poiche' intendiamo eseguire schedule() ed il cambio di contesto solo se si tornera' ad eseguire un job, e' necessario determinare se si stanno gestendo una o piu' interruzioni annidate
9.2.1 Aggiungere una variabile globale in irq.c con il livello di annidamento corrente:

```
unsigned long irq_level = 0ul;
```

9.2.2 Modificare la funzione _bsp_irq() in modo da incrementare irq_level all'ingresso della funzione e decrementarlo all'uscita

```
void _bsp_irq(void) {
    u32 irqno;
    isr_t isr;

    ++irq_level;
    data_sync_barrier();

    for (;;) {
        if (iomem(...)) {
            --irq_level;
            data_sync_barrier();
            return;
        }
    ...
    isr();
    ...
```

9.3 Aggiungere il controllo di irq_level al ritorno da _bsp_irq():

```
    msr spsr_cxsf, r0
    ldr r0, =irq_level
```

<<< gestore da basso livello, dove siamo tornati dal gestore di medio livello.

```
|    ldr r0, [r0]                    | <<<
|    tst r0, r0                      | <<<
|    bne .Lnosched                   | <<<
|[..]                                |
|.Lnosched:                          | <<<
|    ldmfd sp, {r0-r3, r12, lr}^     |
|[..]                                |
+------------------------------------+
```

9.3 Aggiungere il controllo della variabile trigger_schedule:

```
+------------------------------------+
|    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}^     |
|[..]                                |
+------------------------------------+
```

controllo se trigger_schedule è a 0. prendendo il valore e facendo tst r0,r0 (ovvero se r0 vale r0). se si (beq condition) vado in nosched.

9.4 Forza il ritorno in SYSTEM mode alla etichetta _irq_schedule:

```
+------------------------------------+
|    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                   |
|    msr spsr_c, #(SYS_MODE|NO_IRQ)  | <<<
|    ldr lr, =_irq_schedule          | <<<
|    movs pc, lr                     | <<<
|.Lnosched:                          |
|    ldmfd sp, {r0-r3, r12, lr}^     |
|[..]                                |
+------------------------------------+
```

prima abbiamo visto le condizioni per non eseguire schedule, ovvero:
- irq_level, se != 0 salto a Lnosched.
- trigger schedule, se 0 salto a Lnosched.

Adesso, in lr carico l'indirizzo di irq_schedule, per chiamarlo e fare movs pc,lr

9.5 Definire l'invocazione di schedule() in _irq_schedule:

```
+------------------------------------+
|_irq_schedule:                      |
|    sub sp, sp, #32                 |
|    ldr r12, =schedule              |
|    mov lr, pc                      |
|    bx r12                          |
+------------------------------------+
```

alloco spazio su stack, mi salvo in r12 indirizzo schedule, metto in lr il pc, e salto a r12.

9.5.1 Lo stack pointer e' aggiustato in modo da preservare i valori dei registri salvati dal gestore a basso livello dell'interruzione

9.5.2 La funzione schedule() e' eseguita con interruzioni disabilitate ed in modalita' SYSTEM

9.5.3 Il valore restituito da schedule() e' nel registro r0

9.6 Controllare il valore restituito da schedule(): se e' NULL non occorre effettuare un cambio di contesto, quindi si riprende con il task che era in esecuzione al momento della interruzione

```
+-----------------------------+
|    tst r0, r0               |
|    beq .Lnoswitch           |
+-----------------------------+
```

9.6.1 L'etichetta .Lnoswitch e' definita piu' avanti

9.7 E' necessario cambiare il contesto di esecuzione: invocare la funzione _switch_to(), passando come parametro nel registro r0 il valore restituito da schedule()

```
+-----------------------------+
|    ldr r12, =_switch_to     |   per quando ritorno dalla funzione, riprendo da qui.
|    mov lr, pc               |
|    bx r12                   |
+-----------------------------+
```

9.7.1 Al termine della funzione _switch_to() lo stack e' stato cambiato: il registro r13/sp punta alla cima dello stack del task che deve essere posto in esecuzione

9.8 Recuperare il valore dei registri dal [nuovo] stack e riprendere l'esecuzione del [nuovo] job

```
+-----------------------------------+
|.Lnoswitch:                        |   Mettiamo Q (sullo stack) prima in r0, e dopo nello status register, ovvero cpsr_csxf.
|    ldr r0,[sp, #(7*4)]            |   Poi mettiamo nello stack i valori salvati, e aggiungiamo 32 al valore di sp, che è l'operazione di pop.
|    msr cpsr_csxf, r0              |
|    ldmfd sp!, {r0-r3, r12, lr}    |   Nell'ultima riga ritorno a "x" che è lo stato iniziale. (vedi slide e10.27 per riferimento visivo).
|    ldr pc,[sp], #(2*4)            |
+-----------------------------------+
```

9.8.1 Nel caso si fosse saltato qui senza eseguire _switch_to(), il job ripristinato e' quello che era in esecuzione prima dell'interruzione

9.8.2 L'ultima istruzione:

9.8.2.1 Carica in pc/r15 l'indirizzo di ritorno che e' sullla cima dello stack

9.8.2.2 Incrementa sp per rimuovere indirizzo di ritorno e i flag dalla cima dello stack

10 Scrivere la funzione _sys_schedule(), che invoca lo scheduler quando un job termina

10.1 Lo stack non contiene i valori salvati dal gestore delle interruzioni

10.1.1 E' necessario salvare i valori dei registri AAPCS-clobbered sullo stack, ma senza avere a disposizione i registri della modalita' IRQ

10.2 Salvare il valore di ritorno contenuto nel registro r14/lr sullo stack (punta all'interno del ciclo in task_entry_point())

```
+-----------------------------+
|    str lr, [sp, #-(4*2)]!   |
+-----------------------------+
```

10.2.1 Notare il "!" che forza l'aggiornamento di r13/sp

10.2.2 Sottraendo 8 al registro r13/sp si lascia lo spazio per salvare il contenuto del registro cpsr (non si puo' salvarlo prima perche' non ci sono registri su cui operare)

10.3 Salvare il valore del registro cpsr nella locazione lasciata libera:

```
mrs     lr, cpsr
str     lr, [sp, #4]
ldr     lr, [sp]
```

10.4 Salvare il valore dei registri AAPCS-clobbered sullo stack:

```
stmfd   sp!, {r0-r3,r12,lr}
```

10.5 Ora la struttura dello stack e' identica a quella che si ha in seguito all'occorrenza di una interruzione: utilizzare il codice della funzione _irq_schedule() per invocare lo scheduler ed, eventualmente, effettuare il cambio di contesto

```
b .Lnosub32
```

10.5.1 L'etichetta .Lnosub32 evita di sommare 32 a sp:

```
_irq_schedule:
   sub sp, sp, #32
.Lnosub32:
   ldr r12, =schedule
   mov lr, pc
```

10.6 Aggiungere in comm.h la definizione per _sys_schedule()

11 Modificare la funzione main() per rimuovere l'invocazione di run_periodic_tasks()

11.1 E' necessario pero' aggiungere un ciclo senza fine a main(), che costituisce in pratica il programma del task 0:

```
void main(void) {
[..]
    for (;;)
      cpu_wait_for_interrupt();
```

i2 La funzione schedule(), quando invocata viene sempre eseguita con le interruzioni disabilitate.

12.1 Abilitare le interruzioni durante l'esecuzione di select_best_task()

12.2 RACE CONDITION: schedule() puo' essere interrotta da una altra istanza di se stessa

12.3 Aggiungere una variabile di guardia 'do_not_enter' in schedule():

```
struct task *schedule(void)
{
    static int do_not_enter = 0;
    struct task *best;
    unsigned long oldreleases;
    if (do_not_enter)                    <<<
        return NULL;                     <<<
```

```
   |     do_not_enter = 1;                              <<<   |
   |     do {                                                 |
   |         oldreleases = globalreleases;                    |
   |         irq_enable();                              <<<   |
   |         best = select_best_task();                       |
   |         irq_disable();                             <<<   |
   |     } while (oldreleases != globalreleases);             |
   |     do_not_enter = 0;                              <<<   |
   |     trigger_schedule = 0;                                |
   |     return (best != current ? best : NULL);              |
   | }                                                        |
   +----------------------------------------------------------+
```

```
/*
vim: tabstop=4 softtabstop=4 expandtab list colorcolumn=74 tw=73
*/
```