13/12/2022

SCALETTA LEZIONE SERT 13.12.2022 (E11)

1 Integrazione con lo scheduler a priorita' fissa
  1.1 I task EDF avranno sempre priorita' minore dei task a priorita' fissa
    1.1.1 Motivazione #1: implem. di scheduler ibridi come EDF-US[1/2]
    1.1.2 Motivazione #2: implementazione di CBS
  1.2 Modifiche al descrittore di task per il supporto a EDF (comm.h):
    1.2.1 Campo 'priority': nel caso di task a priorita' fissa (FPR), era la priorita' del task; nel caso di task EDF, indica la priorita' del job in esecuzione o primo job pendente
    1.2.2 Nuovo campo 'deadline': scadenza relativa del task EDF
      1.2.2.1 Vale sempre 0 per task a priorita' fissa

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

  1.3 Modifiche alla funzione create_tasks() in tasks.c:
    1.3.1 Cambio nome argomento: da 'priority' a 'prio_dead' (per indicare che in caso di FPR e' la priorita' statica, mentre in caso di EDF e' la scadenza relativa)
    1.3.2 Aggiunto un parametro intero 'type' per indicare se il task da creare e' FPR oppure EDF
      1.3.2.1 Aggiungere due macro FPR e EDF a comm.h:

```
#define FPR 0
#define EDF 1
```

      1.3.2.2 Modificare il prototipo di create_task() in comm.h
    1.3.3 Modificare l'inizializzazione del campo 'priority':

```
t->releasetime = ticks + delay;
if (type == EDF) {                                  <<<
        if (prio_dead == 0)                         <<<
                return -1;                          <<<
        t->priority = prio_dead + t->releasetime;   <<<
        t->deadline = prio_dead;                    <<<
} else { //Fixed Priority                           <<<
        t->priority = prio_dead;                    <<<
        t->deadline = 0;                            <<<
}                                                   <<<
```

  1.4 Modifiche alla funzione select_best_task() in sched.c:
    1.4.1 Aggiungiamo due flag 'edf' e 'fpr': rimangono a zero finche'

non viene trovato un job pendente EDF o FPR, rispettivamente

1.4.2 Il controllo della priorita' dipende dal valore di 'edf':

```
edf = fpr = 0;
[...]
if (fpr) { /* there are pending FPR jobs */
    if (f->deadline != 0)
        continue; /* an EDF job cannot win */
    if (f->priority < maxprio) {
        maxprio = f->priority;
        best = f; /* replace FPR champion */
    }
    continue;
}
/* still no pending FPR jobs */
if (f->deadline == 0) {
    fpr = 1; /* this is the first FPR job */
    maxprio = f->priority;
    best = f;
    continue;
}
/* the pending job is EDF, and no FPR pending
    jobs found up to now */
if (!edf || time_before(f->priority, maxprio)) {
    edf = 1;
    maxprio = f->priority;
    best = f; /* replace EDF champion */
}
```

1.5 Aggiornamento della priorita' dei job EDF

1.5.1 Non puo' essere fatto in check_periodic_tasks() perche' il campo 'priority' del descrittore di task e' unico e fa riferimento al job non ancora completato

1.5.2 Per il primo job di un task nessun problema: la priorita' corretta e' impostata da create_task()

1.5.2 Modifichiamo la funzione task_entry_point() in tasks.c per

1.5.2.1 controllare il rispetto della deadline

1.5.2.2 aggiornare il campo 'priority' per il prossimo job

```
        --t->released;
>>>     if (t->deadline != 0) {
>>>         if (time_after(ticks, t->priority))
>>>             printf("EDF task '%s': deadline miss!\n",
>>>                    t->name);
>>>         t->priority += t->period;
>>>     }
        _sys_schedule();
```

1.5.2.3 Poiche' in create_task() si e' inizializzato il valore della scadenza assoluta come

t->priority = t->releasetime + prio_dead

la prossima scadenza assoluta dovra' essere necessariamente a distanza di un periodo

1.6 Test della schedulazione EDF

1.6.1 Modificare le invocazioni di create_task in main() per

aggiungere il tipo 'EDF'

1.6.2 Creare un task EDF molto lungo in main.c:

```
static void very_long_job(void *arg)
{
   unsigned long now = ticks + HZ*50;
   arg = arg;
   while (time_before(ticks, now)) {
      printf("%8u\r", now-ticks);
      cpu_wait_for_interrupt();
   }
}
[...]
if (create_task(very_long_job, NULL, 60*HZ, 100,60*HZ, EDF,
                "very_long_job") == -1) {
    puts("ERROR: cannot create task very_long_job\n");
    panic1();
}
```

1.6.2.1 Verificare il comportamento

1.6.2.2 Cambiare il task da EDF a FPR e verificare il comportamento

1.7 Creare un task per il controllo del 'watchdog' hardware timer

1.7.1 Consultare il manuale del chipset AM-335x (section 20.4)

1.7.1.1 In sintesi: scrivere un valore diverso da quello corrente nel registro WDT_WTGR ricarica il contatore

1.7.2 Scrivere il file bbb_watchdog.h:

```
#define WDT1_BASE 0x44e35000
iomemdef(WDT1_WTGR, WDT1_BASE + 0x30);
```

1.7.3 Scrivere nel file watchdog.c la funzione rearm_watchdog():

```
static void rearm_watchdog(void *arg)
{
        arg = arg;
        iomem(WDT1_WTGR)++;
}
```

1.7.4 Scrivere nel file watchdog.c la funzione init_watchdog():

```
#define WDT_Ticks   (HZ*30)
[...]
void init_watchdog(void)
{
    if (create_task(rearm_watchdog, NULL, WDT_Ticks, 1,
                WDT_Ticks, FPR, "watchdog") == -1) {
        puts("ERROR: cannot create task \"watchdog\"\n");
        panic0();
    }
}
```

1.7.5 Aggiungere invocazione di init_watchdog() in _init()

1.7.6 Test del corretto funzionamento

2 Implementazione dell'"idle" task

2.1 Aggiungere la funzione idle_task() in main.c

```
static void idle_task(void)
{
 for (;;)
     cpu_wait_for_interrupt();
}
```

2.1.1 Invocare idle_task() come ultima operazione di main()

3 Utilizzo del LED 3 per segnalare l'attivita' della CPU

3.1 Trasformare la funzione led_cycle() in heartbeat():

```
static void heartbeat(void *argv __attribute__ ((unused)))
{
  static int state = 0;
  if (state)
      led0_on();
  else
      led0_off();
  state = 1 - state;
}
```

3.2 Modificare in main() la creazione del task:

```
if (create_task(heartbeat,NULL,HZ,HZ,HZ,EDF,"heartbeat") == -1) {
    puts("ERROR: cannot create task heartbeat\n");
    panic1();
}
```

3.3 In idle_task(), spegnere il LED 3 prima di sospendere la CPU:

```
 for (;;) {
     led3_off();
     cpu_wait_for_interrupt();
 }
```

3.4 In schedule(), modificare in modo da accendere il LED 3 ogni volta che si forza un cambio di contesto

```
trigger_schedule = 0;
if (best == current)                    <<<
    best = NULL;                        <<<
else                                    <<<
    led3_on();                          <<<
do_not_enter = 0;
```

3.4.1 Se la CPU sta per eseguire un task diverso da idle, il LED restera' accceso

3.4.2 Altrimenti il LED verra' spento da idle_task()

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