Università degli Studi di Firenze

Ingegneria Elettrica e dell'Automazione - Curriculum Automazione e Robotica

---

Elaborato di Software Engineering For Embedded Systems

Modellazione e implementazione di taskset in Linux RTAI

Studente
Marco Minarelli

Anno Accademico 2022/2023

Obiettivi

1. Esplorare le API offerte da Linux RTAI;
2. Implementazione di una funzione di busy sleep;
3. Modellare un task-set su Oris, scrivere il codice tramite le API di RTAI e testarlo sull'hardware di riferimento.

Linux RTAI

Real Time Application Interface

Servizi del kernel Linux + peculiarità sistema operativo Real Time

Task sia soft- che hard-real time

FIFO e Round Robin

Hardware di riferimento

Intel NUC NUC10i3FNHN

Busy Sleep

Busy Sleep

Funzione che simula l'utilizzo della CPU per un certo periodo di tempo, passato come input

Algoritmo 1 Struttura di busy_sleep()

Require: $ex\_time > 0$ l'execution time desiderato

$n\_cycles = f(ex\_time)$

$var = 0$

for $i = 0, \ldots, n\_cycles$ **do**

$var = var + 1$

end for

Busy Sleep - Assunzioni

1. Le due grandezze sono legate in modo lineare
2. Esiste sempre un overhead non nullo

Busy Sleep - Assunzioni

1. Le due grandezze sono legate in modo lineare
2. Esiste sempre un overhead non nullo

$$ex\_time = m \cdot n\_cycles + q \tag{1}$$

Regressione Lineare

La regressione formalizza (e risolve) il problema di una relazione funzionale tra variabili misurate.
Si suppone di conoscere alcune coppie di valori $(x_i, y_i)_{i=1}^{n}$ e si suppone che valga

$$y_i = m \cdot x_i + q \tag{2}$$

$$m = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2} \tag{3}$$

$$q = \frac{\sum y - m\sum x}{n} \tag{4}$$

Busy Sleep - Calcolo dei valori

Algoritmo 2 Struttura del singolo test

Require: *n_cycles* il numero di iterazioni da eseguire
Require: *n_test* il numero di test da eseguire
for $k = 1, \ldots, n\_test$ **do**
$t_1 = time()$
$var = 0$
for $i = 0, \ldots, n\_cycles$ **do**
$var = i + 1$
end for
$t_2 = time()$
$y_k = t_2 - t_1$
end for
Calcola la media dei *n_test* test e ritorna

Busy Sleep - Calcolo dei valori

Algoritmo 3 Struttura dei test

Require: *n_max* massimo numero di iterazioni da seguire
Require: *step* passo tra un'iterazione e l'altra
Require: *n_test* il numero di test da eseguire
  for $k = 0, \ldots, n\_max$ con passo *step* **do**
    $(x_k, y_k) = (k, single\_test(k, n\_test))$
  end for
  ritorna $(x_k, y_k)_{k=0}^{n\_data}$

Busy Sleep - Codice

```
void busy_sleep(RTIME time_ns){
        long n_cycles = (time_ns - q) / m;
        int dummy = 0;
        int i;
        for(i = 0; i < n_cycles; i++){
                dummy = dummy + 1;
        }
}
```

Busy Sleep - Risultati

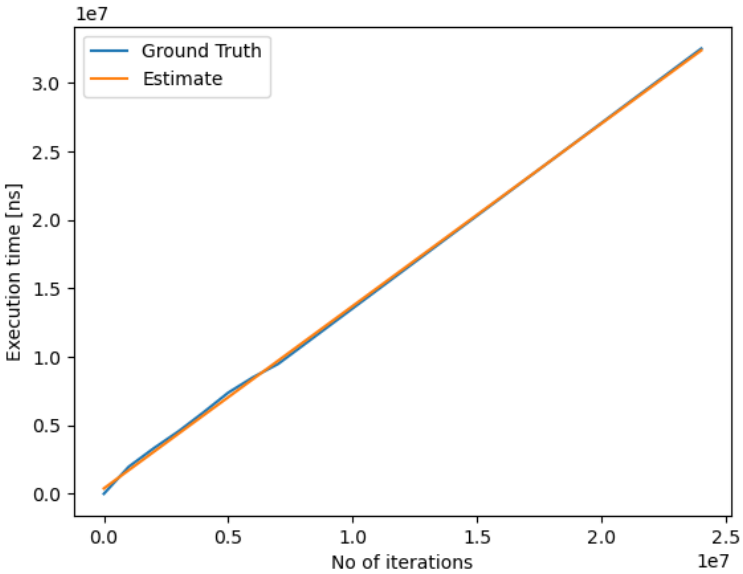


Figura: Confronto tra i dati e la retta interpolata

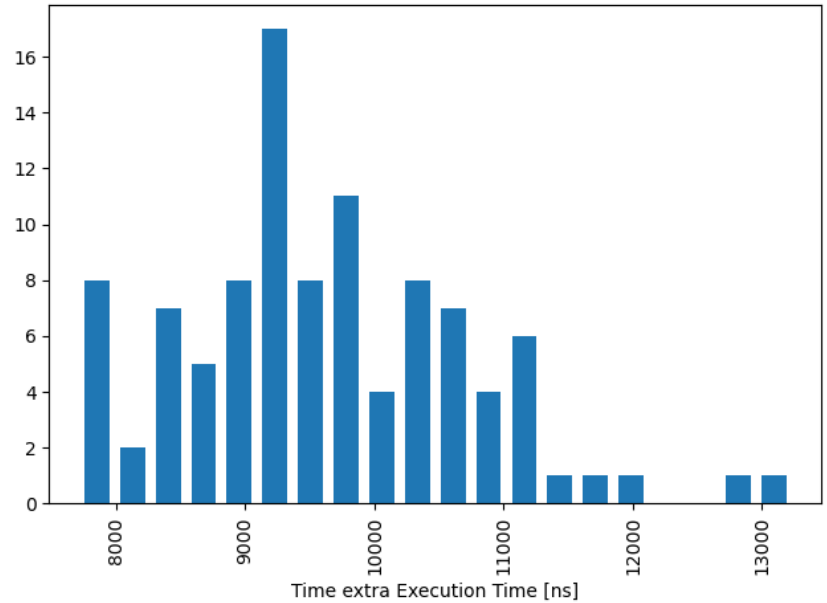


Figura: Anticipo della busy sleep rispetto ad un tempo di 20 millisecondi

Taskset

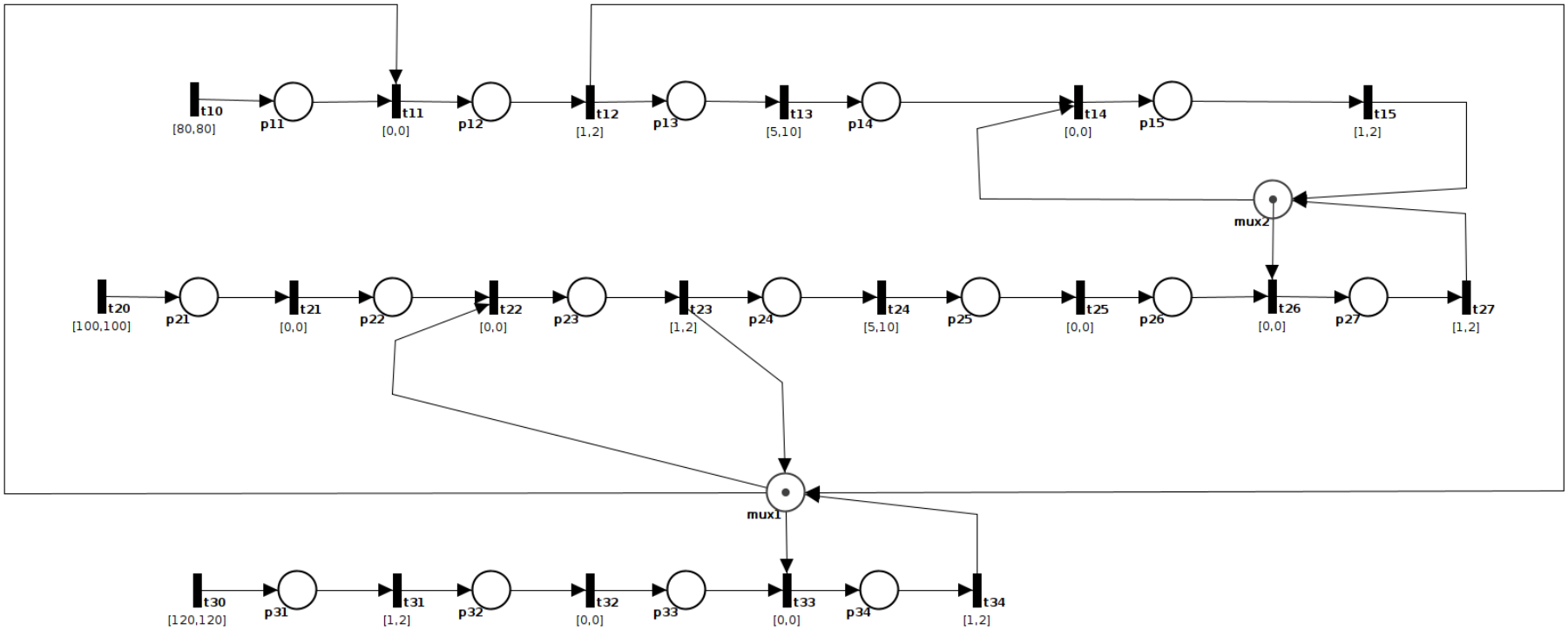


Figura: Modellazione del taskset

Codice

`rand()`

Codice

rand()

```
unsigned rand(unsigned min, unsigned max){
        unsigned random_val;
        get_random_bytes(&random_val, sizeof(unsigned));
        random_val = min + (random_val % (max + 1 - min));
        return random_val;
}
```

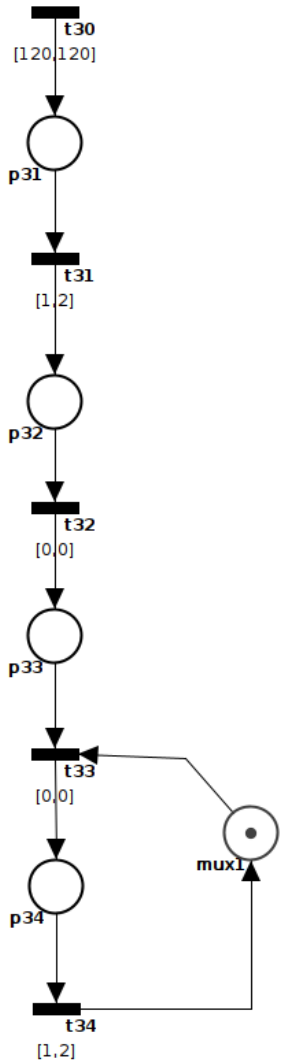
Codice

`tskN_job()`

Codice

tskN_job()

```
void tsk3_job(long arg){
  while(1){
    send_log(3, 30);
    busy_sleep(rand(1, 2) * MILLISEC);
    send_log(3, 31);
    rt_change_prio(rt_whoami(), 1);
    send_log(3, 32);
    rt_sem_wait(&mux1);
    send_log(3, 33);
    busy_sleep(rand(1, 2) * MILLISEC);
    rt_change_prio(rt_whoami(), 3);
    rt_sem_signal(&mux1);
    send_log(3, 34);
    rt_task_wait_period();
  }
```



Codice

`init_module()`

Codice

init_module()

```
int init_module(void){
  ...
  res = rtf_create(FIFO_ID_LOG, sizeof(log)*(LOG_VAL));
  calibrate_busysleep(MAX_VAL, STEP, 3);
  rt_typed_sem_init(&mux1, 1, BIN_SEM);
  ...
  rt_task_init(&tsk3, (void*)tsk3_job, 0, 10000, 3, 0, 0);
  ...
  rt_task_make_periodic(&tsk3, start3, nano2count(120*
      ↪MILLISEC));
  ...
  return 0;
}
```

Codice

`send_log()`

Codice

send_log()

```
void send_log(int task_id, int trans_id){
        log l;
        l.task_id = task_id;
        l.transition_id = trans_id;
        l.time = rt_get_cpu_time_ns();
        rtf_put(FIFO_ID_LOG, &l, sizeof(l) );
}
```

Log

Listato 1: Struttura del log

```
[transition name]
[time to fire]
[transition name]
[time to fire]
```

Listato 2: Esempio di log

```
t10
80
t11
0
t12
2
t13
10
t14
0
t15
2
...
```

Possibili sviluppi

- Cambiare l'hardware di riferimento;
- Modificare il task set;
- Modificare gli algoritmi di scheduling impiegati.

Conclusioni

- La busy sleep è stata implementata per l'hardware proposto
- Il task set modellato grazie a Oris è stato scritto con le API di Linux RTAI
- Il log ottenuto dalla sua esecuzione è stato dato in pasto al tool per la verifica con esito positivo

Università degli Studi di Firenze

Ingegneria Elettrica e dell'Automazione - Curriculum Automazione e Robotica

---

Elaborato di Software Engineering For Embedded Systems

Modellazione e implementazione di taskset in Linux RTAI

Studente
Marco Minarelli

Anno Accademico 2022/2023