Sistemi in Tempo Reale
Compito del 15 giugno 2009

Esercizio 1

Un sistema real-time consiste di una catena di task dai task τ_1, τ_2, τ_3, τ_4. Il task τ_1 è periodico di periodo $T_1 = 50$msec, gli altri vengono attivati nella maniera seguente: τ_2 e τ_3 vengono attivati al termine di τ_1, mentre τ_4 viene attivsato quando sia τ_2 che τ_3 hanno completato. Inoltre, la catena ha una deadline di 90msec (ovvero il task τ_4 deve completare al massimo 90 msec da quando il task τ_1 è stato attivato).

1. Scrivere il codice dei 4 task, e le strutture dati per le sincronizzazione fra thread, in modo che il sistema sia in grado di riconoscere se la deadline del task $\tau_4 4$ viene mancata.

2. Supponendo che i tempi di calcolo e le priorità siano assegnate come nella tabella in basso, e che ci siano altri 2 task periodici, τ_5 e τ_6 nel sistema, calcolare il response time del task τ_4.

Task	C_i	T_i	p_i
τ_1	10	50	1
τ_2	5	–	2
τ_3	7	–	3
τ_4	6	–	4
τ_5	3	20	6
τ_6	5	40	5

Esercizio 2

Calcolare il tempo di bloccaggio con priority inheritance del seguente insieme di task (ordinati per livello di preemption decrescente). Disegnare un esempio di situazione di bloccaggio per il task τ_3 nel caso di schedulazione a priorità fisse.

Task	S_1	S_2	S_3
τ_1	1	0	0
τ_2	0	2	3
τ_3	0	0	0
τ_4	4	2	2
τ_5	2	3	3

Soluzione esercizio 1

Per prima cosa, progettiamo un meccanismo per sincronizzare i task l'uno con l'altro, e permettere il passaggio dell'informazione sull'istante di attivazione.

```
class SynchObj {
  pthread_mutex_t m;
  pthread_cond_t c;
  int signals;
  int activations;
  int blocked;
  struct timespec last;
public:
  SynchObj(int n);
  void activate(struct timespect *act);
  struct timespec wait();
};

SynchObj::SynchObj(int n) : signals(n), activations(0), blocked(0)
{
  pthread_mutex_init(&m, 0);
  pthread_cond_init(&c, 0);
}

void SynchObj::activate(struct timespec *act)
{
  pthread_mutex_lock(&m);
  last = *act;
  activations++;
  if (activations >= signals && blocked) {
    pthread_cond_signal(&c);
    activations -= signals;
    blocked = 0;
  }
  pthread_mutex_unlock(&m);
}

struct timespec SynchObj::wait()
{
  pthread_mutex_lock(&m);
  if (activations >= signals)
    activations -= signals;
  else {
    blocked = 1;
    pthread_cond_wait(&c, &m);
  }
  return last;
  pthread_mutex_unlock(&m);
}
```

Vengono preparati 3 di questi oggetti, uno per ciascuno dei task τ_2, τ_3 e τ_4.

```
SynchObj s2(1), s3(1), s4(2);
```

A questo punto, il codice del thread τ_1 è il seguente:

```
void *task1(void *arg)
{
  int period = 50000;     // 50 msec
  struct timespec *next;

  clock_gettime(CLOCK_REALTIME, &next);
  while (1) {
    // codice
    s2.activate(&next);
    s3.activate(&next);
    timespec_add_us(&next, period);
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &next, NULL);
  }
}
```

Per il task τ_2:

```
void *task2(void *arg)
{
  struct timespec act;
  while (1) {
    act = s2.wait();
    // codice
    s4.activate(&act);
  }
}
```

Per il task 3 è analogo, basta sostituire `s2` con `s3`. Infine, per il task τ_4,

```
void *task4(void *arg)
{
  struct timespec act;
  struct timespec finishing;
  int deadline = 900000; // 90 msec
  while(1) {
    act = s4.wait();
    // codice
    clock_gettime(CLOCK_REALTIME, &finishing);
    timespec_add_us(&act, deadline);
    if (timespec_cmp(&finishing, &act) > 0) {
      printf("DEADLINE␣MISSED!\n");
  }
}
```

Riguardo al calcolo del tempo di risposta di τ_4, possiamo considerare la catena $\tau_1 - \tau_4$ come un unico task τ_x a priorità più bassa rispetto ai task τ_5 e τ_6. Quindi, il task set si può semplificare nel modo espresso dalla seguente tabella.

Quindi, applicando la formula, il tempo di risposta risulta pari a 38.

Task	C_i	T_i	p_i
τ_5	3	20	6
τ_6	5	40	5
τ_x	28	50	1

Soluzione esercizio 2

I tempi di bloccaggio con priority inheritance sono: $B_1 = 4$, $B_2 = 7$, $B_3 = 7$, $B_4 = 3$, $B_5 = 0$. Con fixed priority, un esempio di bloccaggio potrebbe essere il seguente.