Informatica e Sistemi in Tempo Reale
Compito del 6 luglio 2011

Esercizio 1

Un sistema real-time è composto da due modi di funzionamento. I task nel sistema, e i relativi modi di funzionamento sono mostrati in tabella.
Inoltre, il sistema prevede un task manager dal tempo di calcolo trascurabile che si occupa di far cambiare modo al sistema.
I sistema è schedulato da Fixed priority, ed inizialmente si trova nel modo A. Al presentarsi di un evento esterno X, il task manager fa transire il sistema nel modo B.

Task	C	T	Modo
τ_1	1	5	A, B
τ_2	2	8	A
τ_3	3	9	B
τ_4	3	12	A
τ_5	2	10	B
τ_6	3	21	A,B

- Scrivere lo pseudo codice dei task $\tau_2, \ldots, \tau_5$ (basta scrivere un codice template per tutti i task)
- Calcolare il massimo tempo di transizione (da quando è scattato l'evento a quando i task τ_2, τ_4 sono stati disattivati e i task τ_3 e τ_5 sono stati attivati) usando l'idle time protocol.
- Scrivere lo pseudo-codice del task manager.

Esercizio 2

Considerare il seguente insieme di task schedulati con DM.

- Verificare la schedulabilità
- Dire di quanto si può aumentare (o di quanto si deve diminuire) il tempo di calcolo del task τ_2 mantenendo il sistema schedulabile.

Task	C	T	D
τ_1	2	12	5
τ_2	6	20	20
τ_3	5	25	25
τ_4	6	50	50

Soluzione esercizio 1

Come al solito, mettiamo per prima le strutture dati presenti nel file task.h:

```
#ifndef __TASK_H__
#define __TASK_H__

#define NTASK 6
#define NMODE 2

#include <pthread.h>
#include <semaphore.h>

typedef struct mode {
    int matrix[NTASK][NMODE];
    int curr_mode;
    int dest_mode;
    pthread_mutex_t m;
    sem_t block[NTASK];
} mode_data_t;

typedef struct taskdata {
    int index;
    int period;   //in musec
    int priority; // priorita' del task
    mode_data_t * modes;
} task_data_t;

void *taskbody(void *);

#endif
```

Poi, ecco il codice dei task in task.c:

```
#include <stdio.h>
#include "task.h"
#include "time_utils.h"

void set_priority(int prio)
{
    struct sched_param param;
    param.sched_priority = prio;
    sched_setscheduler(pthread_self(), SCHED_FIFO, &param);
}

int isInMode(task_data_t *data) {
    int flag;
    pthread_mutex_lock(&data->modes->m);
    flag = data->modes->matrix[data->index][data->modes->curr_mode];
    pthread_mutex_unlock(&data->modes->m);
    return flag;
}

struct timespec task_disable(task_data_t *data)
{
    struct timespec now;
    sem_wait(&data->modes->block[data->index]);
    clock_gettime(CLOCK_REALTIME, &now);
    return now;
}

void *taskbody(void *arg)
```

```
{
    task_data_t data = *((task_data_t *)arg);
    struct timespec next;

    set_priority(data.priority);
    clock_gettime(CLOCK_REALTIME, &next);

    while (1) {
        if (!isInMode(&data)) next = task_disable(&data);
        printf("Task␣%d:␣sto␣eseguendo␣%d\n", data.index, data.period);
        timespec_add_us(&next, data.period);
        clock_nanosleep(CLOCK_REALTIME, 0, &next, 0);
    }
}

int check_event(task_data_t *data)
{
    return data->modes->dest_mode;
}

void activate_new_task(task_data_t *data, int n, int o)
{
    int i;
    for (i=0; i<NTASK; i++) {
        if (data->modes->matrix[data->index][o] == 0 &&
            data->modes->matrix[data->index][n] == 1)
            sem_post(&data->modes->block[data->index]);
    }
}

void *taskmanager(void *arg)
{
    task_data_t data = *((task_data_t *)arg);
    struct timespec next;
    int old_mode;

    set_priority(data.priority);
    clock_gettime(CLOCK_REALTIME, &next);

    while(1) {
        pthread_mutex_lock(&data.modes->m);
        old_mode = data.modes->curr_mode;
        int new_mode = check_event(&data);
        if (old_mode != new_mode) {
            data.modes->curr_mode = new_mode;
            set_priority(1);
            activate_new_task(&data, new_mode, old_mode);
        }
        pthread_mutex_unlock(&data.modes->m);

        timespec_add_us(&next, data.period);
        clock_nanosleep(CLOCK_REALTIME, 0, &next, 0);
    }
}
```

E infine, il codice per crearli ed inizializzare il tutto. Il main.c contiene anche il codice per fare il cambio di modo attraverso la pressione di un tasto sulla tastiera:

```
#include <stdio.h>
#include "task.h"
#include <pthread.h>
#include <unistd.h>
```

```
mode_data_t modes = {
    {
        {1,1},
        {1,0},
        {0,1},
        {1,0},
        {0,1},
        {1,1}
    },
    0,
    0,
    PTHREAD_MUTEX_INITIALIZER
};

task_data_t tasks[NTASK] = {
    {0, 100000, 20, &modes},
    {1, 200000, 19, &modes},
    {2, 300000, 18, &modes},
    {3, 400000, 17, &modes},
    {4, 500000, 16, &modes},
    {5, 600000, 15, &modes},
};

int main()
{
    int i;
    pthread_t id[NTASK];
    char c;

    for (i=0; i<NTASK; i++) {
        sem_init(&modes.block[0], 0, 0);
        pthread_create(&id[i], 0, taskbody, &tasks[i]);
    }

    while (1) {
        c = getchar();
        if (c == 'a') modes.dest_mode = 0;
        else if (c == 'b') modes.dest_mode = 1;
        else if (c == 'q') break;
    }

    return 0;
}
```

Ricordo che in sede di esame non è necessario produrre un programma compilabile o eseguibile. Ad esempio, è possibile evitare di specificare i file include da utilizzare, oppure addirittura tralasciare pezzi di codice non essenziali allo svolgimento dell'esercizio. E' però necessario che il codice utilizzato per la sincronizzazione dei task sia corretto.

Soluzione esercizio 2

Cominciamo con il riportare la tabella dei task.

Task	C	T	D
τ_1	2	12	5
τ_2	6	20	20
τ_3	5	25	25
τ_4	6	50	50

Per il task τ_2, i punti da considerare sono: $\{12, 20\}$. Ecco le equazioni

$$\begin{aligned} C_1 + C_2 &\leq 12 \Rightarrow 8 \leq 12 \\ 2C_1 + C_2 &\leq 20 \Rightarrow 10 \leq 20 \end{aligned}$$

Per il task τ_3 i punti sono $\{12, 20, 24, 25\}$. Ecco le equazioni:

$$\begin{aligned} C_1 + C_2 + C_3 &\leq 12 \Rightarrow 13 \nleq 12 \\ 2C_1 + C_2 + C_3 &\leq 20 \Rightarrow 15 < 20 \\ 2C_1 + 2C_2 + C_3 &\leq 24 \\ 3C_1 + 2C_2 + C_3 &\leq 25 \end{aligned}$$

Infine, per il task τ_4, i punti sono $\{12, 20, 24, 25, 36, 40, 48, 50\}$. Riportiamo i coefficienti delle equazioni, i punti e il valore in tabella:

C_1	C_2	C_3	C_4	Punto	Ris.
1	1	1	1	12	19
2	1	1	1	20	21
2	2	1	1	24	27
3	2	1	1	25	29
3	2	2	1	36	34
4	2	2	1	40	–
4	3	2	1	48	–
5	3	2	1	50	–

Il sistema risulta quindi schedulabile. Ora proviamo a vedere cosa succede se lasciamo C_2 come incognita. Per il task τ_2:

$$\begin{aligned} C_1 + C_2 &\leq 12 \Rightarrow C_2 \leq 10 \\ 2C_1 + C_2 &\leq 20 \Rightarrow C_2 \leq 16 \end{aligned}$$

Prendiamo il max, quindi ci teniamo $C_2 \leq 16$. Per il task τ_3:

$$\begin{aligned} C_1 + C_2 + C_3 &\leq 12 \Rightarrow C_2 \leq 5 \\ 2C_1 + C_2 + C_3 &\leq 20 C_2 \leq 11 \\ 2C_1 + 2C_2 + C_3 &\leq 24 C_2 \leq 7.5 \\ 3C_1 + 2C_2 + C_3 &\leq 25 C_2 \leq 7 \end{aligned}$$

Prendiamo il max $C_2 \leq 11$, e tra $C_2 \leq 16$ e $C_2 \leq 11$ ci teniamo il minimo, quindi $C_2 \leq 11$. Infine, per τ_4:

$$\begin{aligned} C_2 &\leq -1 \\ C_2 &\leq 5 \\ C_2 &\leq 4.5 \\ C_2 &\leq 4 \\ C_2 &\leq 7 \\ C_2 &\leq 8 \\ C_2 &\leq 8 \\ C_2 &\leq 8 \end{aligned}$$

Il max è $C_2 \leq 8$. Dobbiamo fare il minimo con il precedente, quindi $C_2 \leq 8$ è il risultato finale. Quindi, il tempo di calcolo di τ_2 si può aumentare di 2 unità.