Scheduling di task real-time in ambiente Linux

Corso di
Progetto e Sviluppo di
Sistemi in Tempo Reale

Marcello Cinque

Scheduling real-time in ambiente Linux

- Sommario della lezione:
  - Primitive e comandi per lo scheduling
  - Utilizzo di threads
  - Suggerimenti per la programmazione real-time
- Riferimenti
  - G. Lipari. “Programming RT systems with pthreads”
  - https://pubs.opengroup.org/onlinepubs/009695399/idx/realtime.html

E’ possibile schedulare task con rate monotonic in Linux?

- Tutto ciò di cui abbiamo bisogno è uno scheduler preemptive a priorità fisse
  - E’ sufficiente assegnare le priorità ai task in modo direttamente proporzionale alle loro frequenze
- Lo scheduler di default di liunux non è a priorità fisse
  - La policy di default, SCHED_OTHER, è basata sull’algoritmo CFS
- *Ma*, Linux include anche le classi di scheduling: SCHED_FIFO e SCHED_RR!

Come assegnare priorità fisse con SCHED_FIFO in Linux?

- Tre approcci possibili
  - Uso delle funzioni disponibili in `<sched.h>`
  - Uso del comando `chrt`
  - Utilizzo delle funzioni pthread (RT-POSIX) per modificare classe di scheduling e priorità prima della creazione del thread
- In tutti e tre i casi, l'utente deve essere root

Uso della funzione set scheduler

```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy, const struct sched_param *param);

struct sched_param {
                ...
                int sched_priority;
                ...
            };
```

- `pid`: l'id del task, 0 se è il task chiamante
- `policy`: può essere SCHED_FIFO, SCHED_RR o SCHED_OTHER
- `param`: usato per settare la priorità tra 0 to 99 (deve essere 0 per SCHED_OTHER)

- Per verificare la nuova scheduling policy e la priorità, utilizzare il comando:

```
ps -eLfc
```

Esempio

```
int main()
{
    struct sched_param sp;
    sp.sched_priority = 11;

    sched_setscheduler(0, SCHED_FIFO, &sp);

    start_periodic_timer(2000000, 5000);

    while(1) {
        wait_next_activation();
        job_body();
    }

    return 0;
}
```

Uso del comando `chrt`

```
chrt [options] priority command
chrt [options] -p priority <pid>
```

- `chrt` imposta o recupera gli attributi di scheduling real-time di un task (un nuovo *command* o un pid esistente)
- Alcune opzioni:
  - `-o, --other`: imposta la policy a SCHED_OTHER
  - `-r, --rr`: imposta la policy a SCHED_RR
  - `-f, --fifo`: imposta la policy a SCHED_FIFO
  - `-p <pid>`: per recuperare (o impostare) le info di scheduling di un task
- Per esempio:
  - `sudo chrt -f 11 ./task` lancia «task» con SCHED_FIFO e priorità 11
  - `sudo chrt -f -p 11 2677` imposta SCHED_FIFO e priorità 11 al task con pid 2677

Utilizzo dei pthreads

- Cosa sono i thread? Che differenza c'è con il processo?
- Nei sistemi UNIX il processo viene creato con la system call `fork` ed è composto da:
  - Uno spazio di indirizzamento: codice, dati globali, stack, heap
  - Un flusso di esecuzione (un thread) che parte eseguendo la funzione `main()`
  - Quindi, quando creato, possiamo dire che un processo «contiene» un thread
- E' poi possibile aggiungere nuovi thread al processo creato con la funzione `pthread_create()`
  - I nuovi thread gireranno nello stesso spazio di indirizzamento del processo chiamante, quindi condivideranno l'area dati
  - Avranno un proprio stack privato
  - Inizieranno ad eseguire la funzione passata come argomento alla funzione `pthread_create`
- Lo scheduler non fa distinzione: scehdula task, siano essi processi o thread

Creazione di un Thread

- **`pthread_create (thread,attr,start_r,arg)`**
  - Crea un nuovo thread e lo rende eseguibile.
  - **`thread`** (output): di tipo `pthread_t`, è un identificatore del thread creato;
  - **`attr`** (input): di tipo `pthread_attr_t`, serve a impostare gli attributi del thread, tra cui la classe di scheduling e la priorità!
  - **`start_r`** (input): puntatore (di tipo `void *`) alla funzione C (***starting routine***) che verrà eseguita una volta che il thread è creato;
    - Firma di una starting routine di esempio: **void * foo (void *)**
  - **`arg`** (input): argomento (di tipo `void *`) che può essere passato alla funzione C (ne va fatto il casting a `void *`).

Terminazione di un Thread

- Un thread può terminare per diversi motivi:
  - La starting routine termina la sua esecuzione;
  - Il thread chiama la `pthread_exit()`;
  - Il thread è cancellato da un altro thread con `pthread_cancel()`;
  - L'intero processo termina.

- **`pthread_exit (status)`**
  - Usata per terminare un thread esplicitamente.
  - Se usata nel programma principale (che potrebbe terminare prima di tutti i thread), gli altri thread continueranno ad eseguire.
  - E' buona norma usarla in tutti i thread.
  - **`status`** (input): indica lo stato di uscita del thread.

Un esempio

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid) // start routine
{
   printf("\n%d: Hello World!\n", threadid);
   pthread_exit(NULL); // terminazione thread
}

int main (int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc, t;
   for(t=0; t<NUM_THREADS; t++){
      printf("Creating thread %d\n", t);

      // creazione
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }
   pthread_exit(NULL); // terminazione main
}
```

Passaggio di parametri

- La `pthread_create` consente di passare un **singolo argomento** di tipo `void *;`
- Per passare **più di un argomento** al thread, si può definire e passare una struct, facendone il casting a `void *:`

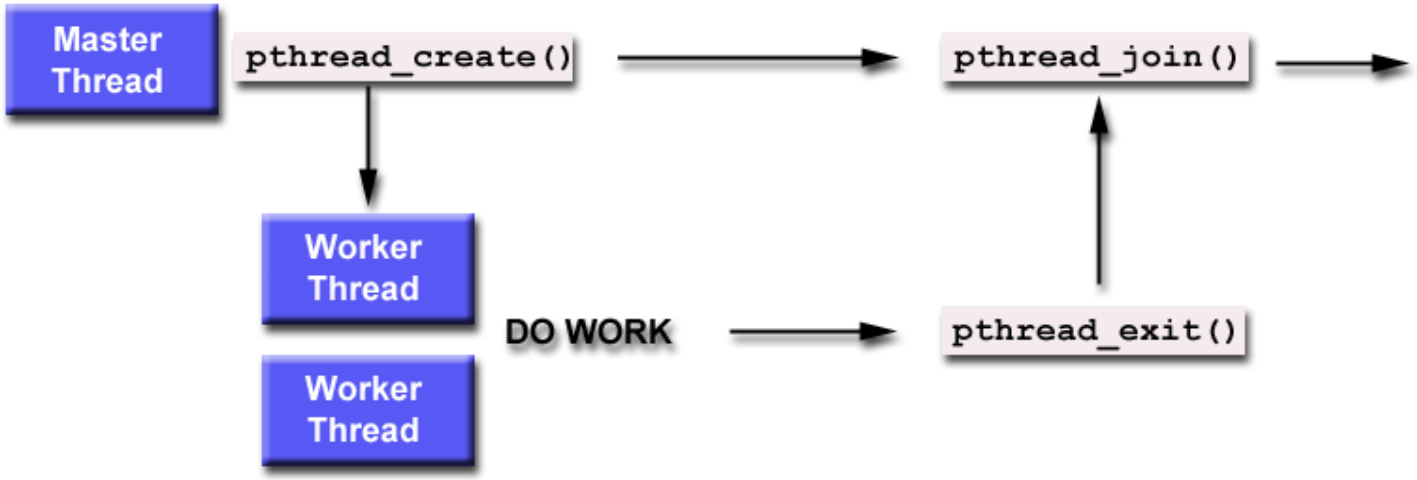
```
struct dati {
    int dato1;
    char dato2;
};
struct dati *d =
            (struct dati *)malloc(sizeof(struct dati));
d->dato1=10; d->dato2='c';
pthread_create(&id, NULL, start_func, (void *) d);
```

- Nel thread sarà sufficiente effettuare il casting inverso:

```
void * start_func( void * d ) {
    struct dati* miei_dati = (struct dati *) d;
    ...
```

Join

- Il Join è un modo per sincronizzare più thread:



- La chiamata **`pthread_join(threadId, status)`** blocca il chiamante finché il thread `threadId` specificato non termina;
- La chiamata consente di ricavare lo stato di uscita del thread (`status`) specificato nella `pthread_exit()`;

Joinable Threads

- Un thread deve essere dichiarato come “joinable” affinché su di esso si possa effettuare l’operazione di join:

```
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
…
pthread_create(&id,&attr, start_r, (void *) data);
…
pthread_join(id, (void **) &status);
```

La join fornisce la struttura dati restituita dal thread mediante `pthread_exit(void *)`

Impostare lo scheduler real-time

- Lo scheduler real-time e la priorità possono essere impostati attraverso una variabile `attr` di tipo `pthread_attr_t`
- Tre impostazioni vanno fatte su `attr`:
  1. Impostare la scheduling policy
  2. Impostare la priorità del thread
  3. forzare lo scheduling esplicito
- `attr` è quindi passato alla `pthread_create` quando si crea il nuovo thread

Impostare la policy

```
#include <pthread.h>

int pthread_attr_setschedpolicy(pthread_attr_t *attr,
                                int policy);
```

- Aggiunge la policy di scheduling all'attributo `attr`
  - `policy`: può essere `SCHED_FIFO`, `SCHED_RR` o `SCHED_OTHER`
- Restituisce 0 se non ci sono errori

Impostare la priorità

```
struct sched_param myparam;
myparam.sched_priority = <value>;
pthread_attr_setschedparam(&attr, &myparam);
```

- La struttura `sched_param` viene utilizzata per impostare la priorità (da 0 a 99) da aggiungere alla variabile `attr` attraverso la funzione `pthread_attr_setschedparam`

Impostare lo scheduling esplicito

- Di default, il thread figlio eredita gli attriubuti di scheduling del thread padre
- Per fare in modo che le impostazioni siano prese in considerazione, è necessario settare lo scheduling come esplicito sulla variabile `attr`

```
int pthread_attr_setinheritsched(pthread_attr_t *attr,
                                 int inheritsched);
```

- `inheritsched` può essere:
  - `PTHREAD_INHERIT_SCHED` (default): I thread creati con questo attributo ereditano le policy del thread genitore; gli attributi impostati nella variabile `attr` sono ignorati.
  - `PTHREAD_EXPLICIT_SCHED`: il thread viene creato utilizzando le impostazioni specificate nella variabile `attr`

Esempio

- Creazione di un FIFO thread con priorità 11

```
pthread_t th;
pthread_attr_t myattr;
struct sched_param myparam;
pthread_attr_init(&myattr);

pthread_attr_setschedpolicy(&myattr, SCHED_FIFO);
myparam.sched_priority = 11;
pthread_attr_setschedparam(&myattr, &myparam);
pthread_attr_setinheritsched(&myattr,PTHREAD_EXPLICIT_SCHED);

pthread_create(&th, &myattr, thread_code, &thread_params);
pthread_attr_destroy(&myattr);
```

Esercizio: scrivi una funzione `pthread_create_fifo` per semplificare l'inizializzazione

Alcuni suggerimenti

- Al fine di evitare comportamenti inattesi, può essere opportuno
  - evitare il ricorso alla paginazione su domanda (evitare page fault)
  - Fare in modo che i processi/thread eseguano su CPU ben determinate (*affinity*), ognuna gestita con una certa policy (ad es, tutti i thread schedulati con rate monotonic in esecuzione sulla stessa CPU)

Evitare page fault

- E’ necessario accedere tutta la memoria necessaria all’inizio e bloccarla (lock) affinchè le pagine non vengano selezionate per lo swap out
  - *Prefaulting*: accedere la memoria durante l’inizializzazione
  - Necessario per lo stack e l’area heap
- Il blocco della memoria (memory lock) può essere ottenuto con la system call `mlockall()` da utilizzare nel main durante l’inizializzazione

Impostare l'affinity

- E' possibile impostare la CPU su cui eseguirà un thread con:

```
int pthread_setaffinity_np(pthread_t thread,
        size_t cpusetsize, const cpu_set_t *cpuset);
```

- Dove:
  - `thread` è il thread di cui impostare l'affinity
  - `cpusetsize` è pari a `sizeof(cpu_set_t)`
  - `cpuset` è l'isieme di CPU su cui può eseguire il thread, settata attraverso funzioni di utilità `CPU_SET`

Impostare l'affinity

- L'affinity può essere impostata anche all'avvio con il comando taskset

```
taskset -c cpus command
```

- Lancia `command` sulle CPU impostate dal flag `-c` (elenco di ID di CPU separato da virgole)
  - Ad es:

```
taskset -c 0,2 ./myProgram
```

  - Lancia `myProgram` sulle CPU 0 e 2.