RIOT OS Round-Robin Scheduling

Thanaphon Leonardi
Università di Genova, DIBRIS
Via All'Opera Pia, 13
S4633433@studenti.unige.it

Dylan Lo Blundo
Università di Genova, DIBRIS
Via All'Opera Pia, 13
S4623731@studenti.unige.it

Premessa

È stato modificato il sistema di scheduling dell'OS, implementando uno scheduler Round Robin con quantum di 0.5 secondi e sostituendo il precedente sistema priority-based.
Le priorità sono state mantenute all'interno della struttura dei thread: questa scelta è stata effettuata per minimizzare i cambiamenti alla struttura originale del sistema operativo.
Inoltre, è stata creata un'applicazione di prova **RRTester** che, in aggiunta ai thread idle e main, genera 5 thread con tempi di servizio differenti.
L'applicazione effettuerà un'operazione di stampa sulla console, segnalando il thread attualmente in esecuzione e calcolando il tempo rimanente al suo completamento.

1 Soluzione Adottata

Inizialmente è stato deciso di introdurre una lista, ordinata cronologicamente, per tenere traccia dei thread (incluso il thread idle).
Dati i requisiti dell'applicazione **RRTester**, la struttura del thread è stata cambiata per tenere anche traccia dei tempi di servizio massimi e rimanenti.
A questo proposito, è stato aggiunto **xtimer**, permettendo sia il calcolo del tempo di servizio rimanente sia lo scorrimento lungo la lista dei thread disponibili all'esecuzione.
Infine, è stata eliminata la runqueue bitcache per non gravare il sistema operativo con carico di lavoro superfluo.

2 Implementazione

2.1 Files Modificati

- *sched.c*
- *sched.h*
- *thread.c*
- *thread.h*

2.2 Applicazione Per Test

---

main.c

L'applicazione **RRTester** inizialmente crea cinque threads, ciascuno con il proprio tempo di servizio. Per simulare un carico di lavoro che necessiterebbe del tempo specificato, i thread sono stati mandati in busy waiting finché non sarà passato il tempo previsto.

2.3 Struttura Del Thread

---

thread.h

struct _thread

```
struct _thread {
    char *sp;                       /**< thread's stack pointer         */
    thread_status_t status;         /**< thread's status                */
    uint8_t priority;               /**< thread's priority              */
    uint64_t service_time;          /**< thread's total executed time   */
    uint64_t max_service_time;      /**< thread's maximum lifetime      */

    kernel_pid_t pid;               /**< thread's process id            */
```

service_time e max_service_time sono stati aggiunti alla struttura dei thread.

2.4 Implementazione Del Thread

thread.c

thread_create()

I necessari cambiamenti sono stati effettuati per mantenere aggiornata la nuova lista dei thread (list_all_threads). Il thread idle, che si comporta diversamente, viene ignorato dalla Round-Robin queue finchè non sarà l'ultimo.

2.5 Algoritmo Di Scheduling

sched.c

sched_run()

```
128     // Thread selection
129     if (active_thread) {
130       if (clist_count(&list_all_threads) == 1) {
131         next_thread = active_thread;
132       } else {
133         next_thread = container_of((&active_thread->rq_entry)->next,
134                                    thread_t, rq_entry);
135       }
136     } else {
137       // Only accept threads in PENDING state
138       next_thread = container_of((list_all_threads.next)->next, thread_t, rq_entry);
139     }
140
141     // Only accept threads in PENDING state
142     if (clist_count(&list_all_threads) > 1) {
143       while (next_thread->status != STATUS_PENDING || next_thread->priority == 15) {
144         next_thread = container_of((&next_thread->rq_entry)->next,
145                                    thread_t, rq_entry);
146
147         if (next_thread->status == STATUS_RUNNING)
148           break;
149       }
150     }
151
152     if (active_thread) {
153         _unschedule(active_thread);
154     }
```

Selezione del thread successivo scorrendo la queue e prelevando il prossimo thread in *pending* (pronto all'esecuzione), incluso opportuni controlli per evitare di chiamare il thread idle qualora ci dovessero essere altri thread nella lista.

2.6 Calcolo Del Service Time

sched.c

sched_run()

```
167       if (service_time_start != 0) {
168         next_thread->service_time += xtimer_now64().ticks64 - service_time_start;
169       }
170
171       if (next_thread->priority != 7) {  // if NOT main thread
172         service_time_start = xtimer_now64().ticks64;
173       }
174
175       sched_active_pid = next_thread->pid;
176       sched_active_thread = (volatile thread_t *)next_thread;
177
178       #ifdef DEVELHELP
179         if (!isIdle) {
180           printf("\nCurrently running thread: Thread %s\n", sched_active_thread->name);
181         }
182
183         if (next_thread->priority == 15) {
184           isIdle = true;
185         } else {
186           isIdle = false;
187         }
188       #endif
189
190       if(next_thread->max_service_time != 0) {
191         if (next_thread->max_service_time > next_thread->service_time) {
192           // Avoid overflow from subtracting unsigned numbers
193           float time = (float) ((next_thread->max_service_time - next_thread->service_time));
194           printf("Thread service time remaining: %.2fs\n", (float) time / 1000000);
195         } else {
196           printf("Thread service time remaining: 0.00s\n");
197         }
198       }
199
```

Calcolo del tempo rimanente; service_time_start rappresenta il momento in cui si è svegliato l'ultimo thread che, sottratto al tempo attuale (xtimer_now64().ticks64), restituisce il tempo rimanente.

Il primo if evita il calcolo quando non è stato ancora assegnato un tempo iniziale, mentre il secondo if permette al main thread di non essere incluso nel calcolo del tempo rimanente.

```
200       xtimer_set(&timer_rr, ROUND_ROBIN_TIME_QUANTUM);
```

Questa funzione è una callback: ogni 0.5 secondi, manda in esecuzione il prossimo thread se disponibile.