Relatório dos Componentes de Escalonamento

Pedro Rosanes

1 de janeiro de 2011

1 Análise do Escalonador Padrão

O escalonador padrão adota uma política FIFO. Ele provê as interfaces *Scheduler* e *TaskBasic*. As tarefas se conectam ao escalonador através da *TaskBasic*. Ao compilar um programa em NesC, todas tarefas básicas viram uma interface desse tipo. Porém, para se diferenciarem é criado um parâmetro na interface [1].

A fila implementada **não** funciona como um vetor circular. Existe um 'ponteiro' para a cabeça e um para o rabo, além de um vetor de tamanho 255 (O que limita a quantidade máxima de tarefas). A cabeça contém o identificador do primeiro na fila. A célula cujo index corresponde ao *id* do primeiro contém o *id* do segundo, e assim em diante. Chega-se ao fim da fila quando a célula contém um identificador de vázio.

2 Análise do Escalonador *Earliest Deadline First*

Este escalonador aceita tarefas com deadline e elege as que tem menor *deadline* para executar. A interface usada para criar esse tipo de tarefas é *TaskDeadline*. O *deadline* é passado por parâmetro pela função *postTask*. As *TaskBasic* também são aceitas como recomendado pelo TEP 106[2].

Em contraste, o escalonador não segue outra recomendação: não elimina a possibilidade de *starvation* pois as tarefas básicas só são atendidas quando não há nenhuma com *deadline* esperando para executar. A fila de prioridades é implementada da mesma forma que a do escalonador padrão1, a única mudança está na inserção. Para inserir, a fila é percorrida do começo até o fim, procurando-se o local exato de inserção. Por tanto o custo de inserir é $\bigcirc(n)$, e o custo de retirar da fila é $\bigcirc(1)$. Uma possível modificação seria utilizar uma *heap*. Mudando o custo de inserção e de retirada para $\bigcirc(\log n)$.

A princípio tive problemas com o componente *Counter32khzC*, pois ele não existe para o *micaz* na simulação. Para poder compilar o escalonador tive de tirá-lo. Ele era usado para calcular a hora atual, e somar ao deadline. Sem esse componente, temos um escalonador de prioridades (mínimo).

[1] Para mais informações sobre interfaces parametrizadas olhar o livro TinyOS Programming[1, s. 8.3 e 9].

3 Configuração de um Escalonador não Padrão

Para substituir o escalonador padrão é preciso colocar uma configuração com o nome *TinySchedulerC* no diretório da aplicação. Dentro desta configuração, amarra-se a interface *Scheduler* a implementação do escalonador. Por exemplo:

```
configuration TinySchedulerC
{
    provides interface Scheduler;
    ...
}
implementation
{
    components SchedulerDeadlineP;
    ...

    Scheduler = SchedulerDeadlineP;
    ...
}
```

É preciso também criar a interface para o novo tipo de tarefa, com o comando *postTask* e o evento *runTask*. Por exemplo:

```
interface TaskDeadline<precision_tag> {
    async command error_t postTask(uint32_t deadline);
    event void runTask();
}
```

Por ultimo, deve-se amarrar a interface da tarefa a do escalonador. Por exemplo:

```
configuration TinySchedulerC
{
    provides interface Scheduler;
    provides interface TaskBasic[uint8_t id];
    provides interface TaskDeadline<TMilli>[uint8_t id];
}
implementation
{
    components SchedulerDeadlineP;
    ...

    Scheduler = SchedulerDeadlineP;
    TaskBasic = Sched;
    TaskDeadline = Sched;
}
```

Para que o escalonador funcione corretamente no simulador é preciso adicionar funções que lidam com eventos no *tossim*. Essas funções foram retiradas do arquivo *opt/tinyos-2.1.1/tos/lib/tossim/SimSchedulerBas*
Primeiro é preciso adicionar ao *Scheduler*:

```
bool sim_scheduler_event_pending = FALSE;
sim_event_t sim_scheduler_event;

int sim_config_task_latency() {return 100;}

void sim_scheduler_submit_event() {
  if (sim_scheduler_event_pending == FALSE) {
    sim_scheduler_event.time = sim_time() + sim_config_task_latency();
    sim_queue_insert(&sim_scheduler_event);
    sim_scheduler_event_pending = TRUE;
  }
}

void sim_scheduler_event_handle(sim_event_t* e) {
  sim_scheduler_event_pending = FALSE;
  if (call Scheduler.runNextTask()) {
    sim_scheduler_submit_event();
  }
}

void sim_scheduler_event_init(sim_event_t* e) {
  e->mote = sim_node();
  e->force = 0;
  e->data = NULL;
  e->handle = sim_scheduler_event_handle;
  e->cleanup = sim_queue_cleanup_none;
}
```

Depois, no *Scheduler.init()* adicione:

```
sim_scheduler_event_pending = FALSE;
sim_scheduler_event_init(&sim_scheduler_event);
```

E por ultimo, no *Scheduler.postTask()*, caso a tarefa tenha sido colocada na fila, adicione:

```
sim_scheduler_submit_event();
```

Referências

[1] P. Levis e D. Gay. TinyOS Programming, capítulo 11, 2009.

[2] P. Levis e C. Sharp. TEP 106: Schedulers and Tasks. `http://www.tinyos.net/tinyos-2.x/doc/html/tep106.html`

[3] Boot Sequence, TinyOS Tutorial. `http://docs.tinyos.net/index.php/Boot_Sequence`