TOSThreads

Pedro Rosanes

19 de abril de 2011

1 Modelo

TOSThreads permite programação em thread no TinyOS, sem violar ou limitar o modelo de concorrência do sistema. O TinyOS executa em uma única thread, a nível de kernel enquanto a aplicação executa em uma ou mais threads, a nível de usuário. Em termos de escalonamento, o kernel tem prioridade máxima, ou seja, a aplicação só executa quando o núcleo do sistema está ocioso. Ele é responsável pelo escalonamento de tarefas e execução das chamadas de sistemas.

A interface entre as threads a nível de usuário e o kernel é feito através de chamadas de sistema bloqueantes. Essas chamadas são implementadas aproveitando os serviços já disponíveis do TinyOS. São responsáveis por manter o estado do serviço *split-phase* que será usado, bloquear a thread que a invocou e acordar a thread do kernel.

Passam a existir três tipos de contextos de execução: tarefas, interrupções e threads. Tarefas e interrupções podem interromper threads de aplicação, mas não o contrário. Threads tem preempção entre elas, de modo que é necessário o uso de primitivas de sincronização. As opções fornecidas são *mutex*, semáforos, barreiras, variáveis de condição, e contador bloqueante. Esta ultima foi desenvolvida especialmente para o TinyOS. Seu uso se dá de forma que a thread fica bloqueada até o contador atingir um número arbitrário, enquanto outras threads podem incrementar ou decrementar esta variável através de uma interface.

O TinyOS retoma o controle sobre a aplicação de dois modos diferentes. No primeiro, uma aplicação faz uma chamada de sistema que posta uma tarefa para processar o serviço. No segundo modo, um manipulador de interrupção posta uma tarefa. Porém, neste caso o TinyOS só acorda depois de terminada a execução da interrupção.

O escalonador de threads utiliza uma política *Round-Robin* com um tempo de 5 milisegundos. É ele que oferece toda a interface para manipulação de threads, como pausar, criar e destruir. É interessante nota que o escalonador não existe em um contexto de execução específico, seu contexto depende de quem utilizou sua interface.

As threads podem ser estáticas ou dinâmicas. A diferença é o momento de criação da pilha e do bloco de controle da thread. O primeiro tipo em tempo de compilação, o segundo em tempo de execução. O bloco de controle, também chamado de *Thread Control Block* (TCB) contém

informações essenciais da thread, como seu identificador, seu estado de execução, o valor dos registradores (para troca de contexto), entre outras.

A troca de contexto é feita por códigos específicos para cada plataforma. É utilizada a linguagem assembly junto com C, para armazenar o valor dos registradores importantes na TCB.

2 Implementação

Detalhes da implementação do *TOSThread*.

2.1 Organização dos diretórios

O diretório raiz do *TOSThread* é */opt/tinyos-2.1.1/tos/lib/tosthreads/*. A estrutura básica de diretórios e suas descrições[1]:

chips: Código específico de chips.

interfaces: Interfaces do sistema.

lib: Extensões e subsistemas.

- **net:** Protocolos de rede (protocolos *multihop*).
- **printf:** Imprime pequenas mensagens através da porta serial (para depuração).
- **serial:** Comunicação serial.

platforms: Código específico de plataformas.

sensorboards: Drivers para placas de sensoreamento.

system: Componentes do sistema.

types: Tipos de dado do sistema (arquivos header).

2.2 Tipos e estruturas de dados

O arquivo **types/thread.h** contém os tipos de dados e constantes excenciais para threads. Estados que um thread pode assumir, como ativo, inativo, pronto e suspenso.

```
enum {
    TOSTHREAD_STATE_INACTIVE = 0, //This thread is inactive and cannot be run
        until started
    TOSTHREAD_STATE_ACTIVE = 1, //This thread is currently running on the cpu
    TOSTHREAD_STATE_READY = 2, //This thread is not currently running, but is not
        blocked and has work to do
    TOSTHREAD_STATE_SUSPENDED = 3, //This thread has been suspended by a system
        call (i.e. blocked)
};
```

[1]Todos os arquivos serão referenciados a partir do diretório raiz */opt/tinyos-2.1.1/tos/lib/tosthreads/*. i.e. *types/thread.h*

Constantes que controlam a quantidade máxima de threads, e o periodo de preempção. Estrutura do thread que contém dados como identificador, ponteiro para pilha, estado, ponteiro para função, registradores.

```
struct thread {
volatile struct thread* next_thread;
    //Pointer to next thread for use in queues when blocked
thread_id_t id;
    //id of this thread for use by the thread scheduler
init_block_t* init_block;
    //Pointer to an initialization block from which this thread was spawned
stack_ptr_t stack_ptr;
    //Pointer to this threads stack
volatile uint8_t state;
    //Current state the thread is in
volatile uint8_t mutex_count;
    //A reference count of the number of mutexes held by this thread
uint8_t joinedOnMe[(TOSTHREAD_MAX_NUM_THREADS - 1)
    / 8 + 1]; //Bitmask of threads waiting for me to finish
void (*start_ptr)(void*);
    //Pointer to the start function of this thread
void* start_arg_ptr;
    //Pointer to the argument passed as a parameter to the start function of this
        thread
syscall_t* syscall;
    //Pointer to an instance of a system call
thread_regs_t regs;
    //Contents of the GPRs stored when doing a context switch
};
```

Estrutura para controle de chamadas de sistema. Contém seu identificador, qual thread está executando, ponteiro para função que a implementa.

```
struct syscall {
struct syscall* next_call;
    //Pointer to next system call for use in syscall queues when blocking on them
syscall_id_t id;
    //client id of this system call for the particular syscall_queue within which
        it is being held
thread_t* thread;
    //Pointer back to the thread with which this system call is associated
void (*syscall_ptr)(struct syscall*);
    //Pointer to the the function that actually performs the system call
void* params;
    //Pointer to a set of parameters passed to the system call once it is running
        in task context};
```

Também existe uma estrutura chamada $init_block$

Referências

[1] P. Levis e C. Sharp. TEP 134: The TOSThreads Thread Library. http://www.tinyos.net/tinyos-2.x/doc/html/tep106.html

[2] TOSThreads, TinyOS Tutorial. http://docs.tinyos.net/index.php/Boot_Sequence